REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
24-09-2009	Final Report	17 September 2008 - 17-Sep-09

4. TITLE AND SUBTITLE

Advanced Quantitative Robust Control Engineering: New Solutions for Automatic Loop-Shaping for SISO and MIMO Systems. Part 1: SISO Systems.

5a. CONTRACT NUMBER
FA8655-08-1-3027

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Professor Mario Garcia-Sanz

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Universidad Publica de Navarra
Dept. de Automatica y Computacion
Pamplona 31006
Spain

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
Unit 4515 BOX 14
APO AE 09421

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Grant 08-3027

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking Universidad Publica de Navarra as follows: The project includes the following topics:

1. Study of the State of the Art on QFT automatic loop-shaping.
2. Analysis of the classical theorems that define the control limitations on performance in SISO systems.
3. Synthesis of the main characteristics of an experienced loop-shaping controller design.
4. From the model uncertainty and the performance and stability specifications, definition of the bound restrictions as quadratic inequalities.
5. Selection of an optimum searching algorithm.
6. Design of the new automatic loop-shaping methodology to design QFT controllers.
7. Validation with several demanding examples.

OUTPUTS:
1. Technical Note TN1 that presents the new automatic loop-shaping QFT methodology, including the new theory developed and the validation cases.
2. Matlab modules and scripts.

15. SUBJECT TERMS
EOARD, Control System, Quantitative Feedback Theory (QFT)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			SURYA SURAMPUDI
UNCLAS	UNCLAS	UNCLAS	UL	122	19b. TELEPHONE NUMBER *(Include area code)* +44 (0)1895 616021

Advanced Quantitative Robust Control Engineering: New Solutions for Automatic Loop-Shaping for SISO and MIMO Systems. Part 1: SISO Systems.

Final Report

	Name and Function	Date	Signature
Prepared by	**Prof. Mario Garcia-Sanz (UPNA)** *Professor in Control Systems* *mgsanz@unavarra.es* **Carlos Molins, MSc. (UPNA)** *Research Student*		
Verified by	**Prof. Mario Garcia-Sanz (UPNA)** *Professor in Control Systems*		
Authorized by	**Prof. Mario Garcia-Sanz (UPNA)** *Professor in Control Systems*		

Public University of Navarra (UPNA)
NCAGE number: 617DB; DUNS number: 465566052
31006 Pamplona, Spain

(Final_Report_EOARD-UPNA_Grant-FA8655-08-1-3027.doc)

SUMMARY

This is the Final Report of the research project "*Advanced Quantitative Robust Control Engineering: New Solutions for Automatic Loop-Shaping for SISO and MIMO Systems. Part 1: SISO Systems.*" (EOARD Award # FA 8655-08-1-3027; Grant # 083027).

The outcome is a new automatic loop-shaping methodology to design QFT controllers for Single-Input-Single-Output (SISO) systems, which uses two techniques: Evolutionary Algorithms and Genetic Algorithms. The methodology is based on the classical theorems that define the control limitations on performance in SISO systems, on advanced techniques to search for optimum solutions subject to restrictions defined as quadratic inequalities, and on previous experience designing QFT compensators on the Nichols Chart. Using this procedure, it is possible to avoid classical restrictive assumptions, such as rectangular boundary for the templates, pre-specified types of bounds, some controller parameter fixed, available initial controller to start from, etc., as well as restrictive requirements of other well-known optimisation techniques, such as continuity, convexity, or differentiability. The new approach does not need such assumptions, overcoming the disadvantages of the previous automatic procedures, which results in very conservative designs, and is easily implemented on existing CAD tools. In conclusion, it contributes to improve QFT controller design, alleviating much of the manual work required at present.

DOCUMENT CHANGE LOG

Issue/Revision	Date	Modification	Modified pages	Observations
Issue 1.0	1.September.2009			

TABLE OF CONTENTS

1. INTRODUCTION

The Quantitative Feedback Theory (QFT) method ([27-30] and [40]) offers a direct, frequency-domain based design approach for tackling feedback control problems with robust performance objectives. In this approach, the plant dynamics may be described by frequency response data, or by a transfer function with mixed (parametric and non-parametric) uncertainty models. One feature that distinguishes QFT from other frequency-domain methods is its ability to deal directly with uncertainty models and robust performance criteria. This is achieved by translating robust performance specifications and uncertainty models into so-called QFT bounds. These bounds, typically displayed on a Nichols chart-like plot, then serve as a guide for shaping the nominal loop transfer function which involves the manipulation of gain, poles and zeros. This design process is executed efficiently using computer-aided design software, such as the QFT Control Design MATLAB Toolbox ([3]), the AFIT CAD tool ([31], [35] and [36]), Qsyn ([21]) and [30], and is effective for simple problems. Nevertheless, QFT designers are often challenged by such control problems due to a lack of loop-shaping experience, and could benefit from an algorithm that automatically provides a first cut solution to the loop-shaping problem. In addition, an automatic loop-shaping facility would enhance the capabilities of the expert QFT designer. Automatic loop-shaping algorithms have been proposed over the past twenty years and this work reports on a new version.

2. AUTOMATIC LOOP-SHAPING METHODOLOGY

2.1. OVERVIEW

Quantitative Feedback Theory ([27-30] and [40]) is a robust frequency domain control design methodology which has been successfully applied in practical problems from different domains [12-21]. One of the key design steps is loop shaping of the open loop gain function to a set of restrictions given by the design specifications and uncertain model of the plant. Although this step has been traditionally performed by hand, the use of CAD tools ([3], [30], [31], [35], [36]) has made the manual loop shaping much simpler. However, the problem of automatic loop shaping is of enormous interest in practice, since the manual loop shaping can be hard for the non experienced engineer, and thus it has received a considerable attention, especially in the last two decades.

The concept of controller design automation in QFT was introduced by Gera and Horowitz [22] who proposed an iterative procedure based on Bode´s famous gain-phase integral to derive the shape of a nominal loop transfer function. The method, however, needs a rational function approximation to obtain an analytical expression for the loop transfer function, and straight line approximations for the nonlinear QFT bounds. Ballance and Gawthrop [1] simplified the iteration process of this technique, using the QFT Control Design Matlab toolbox [3].

Thompson and Nwokah [38] proposed a method based on nonlinear programming techniques wherein the templates of the uncertain plant are approximated by over bounding rectangles. Such a template approximation leads to over bounding in the constraints derived for the optimization.

Bryant and Halikias [5] addressed the problem of automatic loop shaping using linear programming techniques wherein the QFT bounds are approximated by a series of linear approximations. However, their method also leads to conservatism in describing nonlinear QFT bounds by linear inequalities.

Chait et al. [6] proposed a method based on convexification of the nonconvex closed loop bounds. The QFT design problem in this method is posed in terms of the closed loop complementary sensitivity function. The novelty of this method is that the closed loop nonconvex bounds are transformed into linear inequalities without any conservatism, and then a linear program is solved. However, as pointed by these authors, the shortcoming of the method is that it involves fixing the poles of the closed loop transfer function a priori.

The main drawback of the approaches given above lies in attempting to solve a complicated nonlinear optimization problem using convex or linear programming techniques, which generally leads to conservative designs.

The section 1, based on evolutionary algorithms [10] and the section 2, based on genetic algorithms [11] (as Chen et al. [9]), will try to overcome these difficulties, reformulating the problem as one of parameter optimization of a fixed order controller. Those are stochastic methods, which are based on a probabilistic approach. Both methods are computationally demanding, and it is well known that with genetic and evolutionary algorithms one may obtain a local minimum instead of the global minimum. However, knowing what one wants helps to setup the problem in a way that increases the chance of getting a better solution. Besides, it has the advantages that no initial controller is needed and that it uses the precise values of numerical QFT bounds, which overcomes the problems associated with the approximation of the bounds mentioned in the above techniques.

Finally, Nataraj and Tharewall [33] proposed an interval analysis algorithm to automate the controller synthesis step of QFT. It is based on deterministic interval global optimization techniques and also uses the precise values of numerical QFT bounds. However, the controllers obtained with this method suffer from overdesign over the design frequency range.

2.2. STRUCTURE

2.2.1. Introduction

This work introduces a methodology to design automatically QFT controllers for SISO (single input-single output) plants with model uncertainty. The method generalizes other automatic loop-shaping techniques, avoiding restrictive assumptions about the search space –continuity, existence of derivatives, convexity, pre-specified types of bounds, and other matters–. This

methodology applies the Evolutionary Algorithms and the Genetic Algorithms to search the QFT controller that fulfils the control specifications for the whole set of models with uncertainty. Both strategies have been validated with several benchmarks.

2.2.2. Methodology description

This automatic loop shaping methodology has been applied to the two-degree-of-freedom control system shown in Fig.1. The plant under consideration $P(s)$ is member of a family $\wp$, exhibiting a parametric and a bounded non-parametric uncertainty with the following structure:

$$\wp = \{P(\alpha)[1+\Delta_n]: \alpha \in \Omega, \Delta_n \in \Delta\} \tag{1}$$

where

$$|\Delta_n(j\omega)| < m(\omega) \tag{2}$$

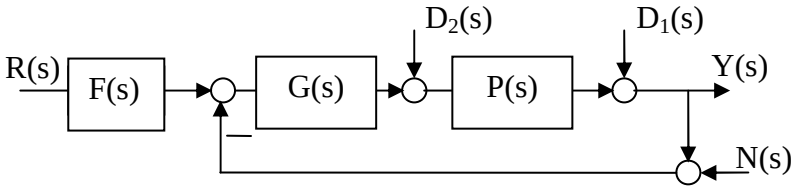

Fig. 1. Two-degree-of-freedom feedback structure with external disturbances and sensor noise signals.

The objective is to design the pre-filter $F(s)$ and the compensator $G(s)$ so that specified robust performance is achieved despite plant uncertainty. The pre-filter can be easily designed following the methodology in Houpis, Rasmussen and Garcia-Sanz [30], and will not be considered here. In this work, performance specifications are not limited to any specific type, such as robust stability, tracking, disturbance rejection, etc. In fact, any type is allowed, as long as it can be numerically computed.

The Fig.2 shows the automatic loop-shaping process proposed in this work. It is comprised of the following steps:

1. - Firstly the designer has to define the system plant and the performance specifications, being then translated into de frequency domain and obtaining the plant templates and the bounds respectively.

On the other hand, the user has to define the search algorithm parameters. Some of these parameters are the next ones:
- Number of individuals that compose the population
- Number of generations
- Crossover and mutation probability
- Fitness function weights …

In addition, the designer has to define the structure of the initial controller to be found. Here the designer will have to define the number of real and/or complex poles and zeros, as well as its range of variation.

2. - Select the strategy to search the optimal controller. In this work the Evolutionary Algorithm and the Genetic Algorithm are applied, but any other strategy may be selected in this step.

3. - Once the search algorithm has provided the best solution, it is saved for further tasks.

4. - Afterwards, the controller obtained with the search algorithm is reduced to a k-th order model, being k the order of the previous controller minus 1. This is performed by means of the optimal Hankel norm approximation [37]. The resulting model is saved as it was done with the previous controller. Then, the process goes back to the step number 2 and calculates the next controller with a structure similar to the one resulted from applying the Hankel norm approximation. This is repeated until the order of the controller is equal to 1.

5. - Finally, with all the saved results some tasks may be performed in order to compare them.

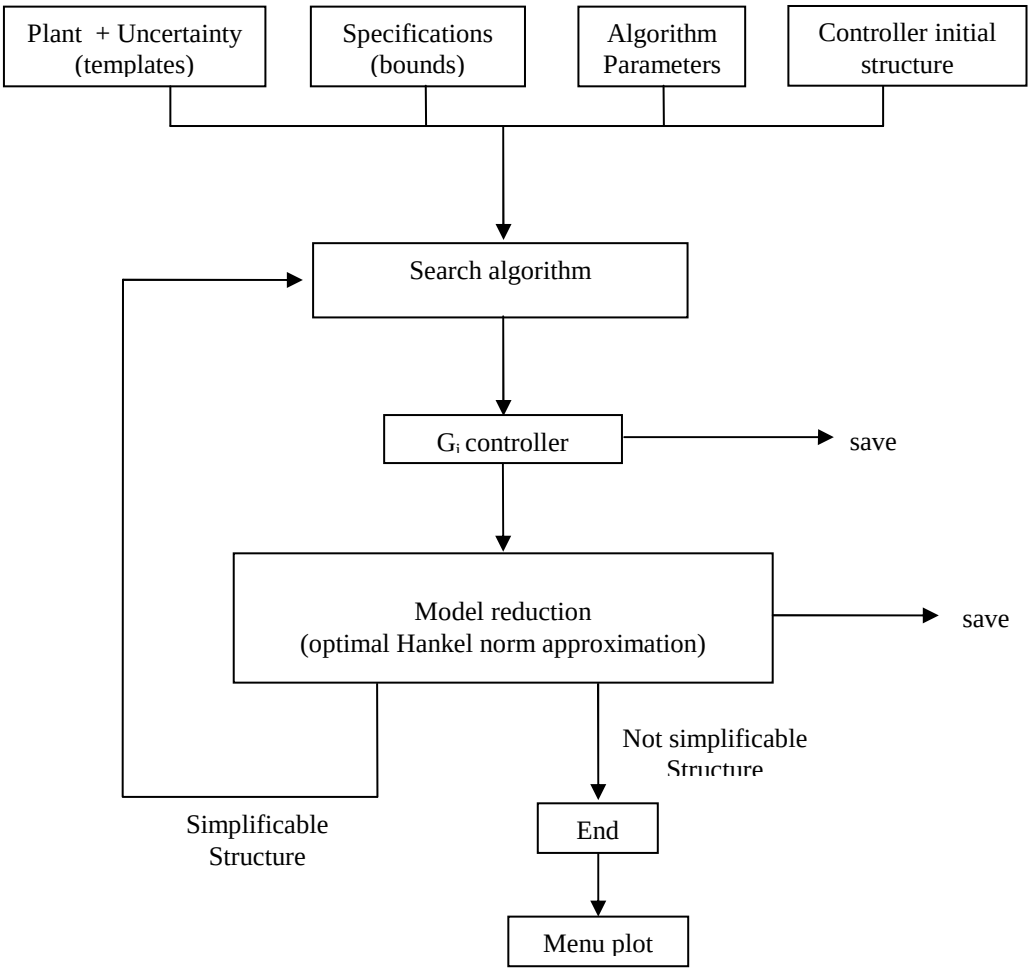

Fig. 2. Methodology diagram

3. AUTOMATIC LOOP-SHAPING OF QFT ROBUST CONTROLLERS VIA EVOLUTIONARY ALGORITHMS

3.1. INTRODUCTION

In this section the methodology a) applies Evolutionary Algorithms to search the QFT controller that fulfils the control specifications for the whole set of models with uncertainty, and b) validates the new approach with a benchmark.

Finally, the Matlab files are described in Appendix A.

3.2. STRATEGY 1: EVOLUTIONARY ALGORITHMS

3.2.1. Problem Statement

In this strategy a general formulation for the controller structure is used. It is expressed by the following transfer function:

$$G(s)=k_G \frac{\prod_{i=1}^{n_{rz}}\left(\frac{s}{z_i}+1\right)\prod_{i=1}^{n_{cz}/2}\left(\frac{s^2}{|z_i|^2}+\frac{2\mathrm{Re}(z_i)}{|z_i|^2}s+1\right)}{s^r\prod_{j=1}^{m_{rp}}\left(\frac{s}{p_j}+1\right)\prod_{j=1}^{m_{cp}/2}\left(\frac{s^2}{|p_j|^2}+\frac{2\mathrm{Re}(p_j)}{|p_j|^2}s+1\right)} \quad (3)$$

where, k_G is the gain, z_i is a zero that may be complex (n_{cz}, number of complex zeros) or real (n_{rz}, number of real zeros), and p_j is a pole (real or complex) with m_{rp} the number of real poles and m_{cp} the number of complex poles. Note that the amount of complex zeros or poles must be even, to have pairs of complex conjugate numbers and obtain a polynomial with real coefficients. The controller may have some poles in the origin and the designer can check parameter r (usually 0, 1 or 2) to set them.

3.2.2. Evolutionary Algorithms

Evolutionary Algorithms (EA) ([23], [32] and [34]) are divided in three major branches: Evolutionary Programming (EP), Evolution Strategies (ES) and Genetic Algorithms (GA). Their common features are the use of a population and the exchange of information among individuals. They apply a progressive evolution of the population to reach a final individual which represents the solution of the problem. These algorithms have been widely applied in engineering and operation research, particularly on numerical optimization problems. These applications usually manage several objectives subject to multiple constraints, so that:

$$\begin{aligned} &\text{Minimize} \quad [f_1(x), f_2(x),..., f_N(x)] \\ &\text{subject to} \quad g_1(x) \le 0, g_2(x) \le 0,..., g_m(x) \le 0 \\ &\qquad x_{i,\min} < x_i < x_{i,\max} \ , \ x = \{x_i\} \end{aligned} \tag{4}$$

In multi-objective optimization [32] the concept of *Pareto-optimal* solution is of great importance. A possible solution x_k is called *Pareto-optimal* if and only if it satisfies the following two conditions:

$$\begin{cases} f_i(x_k) \le f_i(x_q) & \forall i \in \{1,2,...,N\} \\ f_j(x_k) < f_j(x_q) & j \in \{1,2,...,N\} \end{cases} \tag{5}$$

In other words, x_k is *Pareto-optimal* if it has no worse values for all the objectives than any other possible solution x_q, and at least it improves one of the objectives. The problem may have a big number of *Pareto-optimal* solutions, and it could be impossible to find all of them.

Search problems are often established to look for as many solutions as possible. However, this strategy deals with the search of only one *Pareto-optimal* solution. Due to the difficulty of simultaneous optimization of several objectives with some constraints, the multi-objective problem is reduced to a single objective formulation expressed by a fitness evaluation function $J(x)$ with a trade-off among objectives $f_i(x)$ and constraints $g_j(x)$.

$$J(x) = \sum_{i=1}^{N} w_i f_i(x) + \sum_{j=1}^{M} r_j g_j(x) \tag{6}$$

Election of weights (w_i, r_j) is critical to reach a good performance. This is usually the main handicap of the single function approach. Nevertheless, alternatives to a unified fitness function have not, at the moment, clear advantage or they are much more difficult to implement. One of the reasons for the popularity of evolutionary algorithms in multi-objective optimization is their simplicity and capacity to handle with complex fitness functions and non-linear problems. However, although the price one has to pay is a big computing effort in comparison with other techniques, that is perfectly assumed thanks to the speed and capabilities of current computers.

3.2.3. Proposed Algorithm

To study the behaviour of evolutionary algorithms to solve the QFT controller design problem, a simple algorithm based on GA and ES has been developed. The main features that define this algorithm are described in the present section.

1.3.3.1. Coding Scheme

A real coding for the parameters of the individuals in the population has been selected. This is a characteristic shared with the ES, in contrast with the classical GA, which use binary strings.

Individuals collect all controller parameters in a vector.

$$x = \left\lfloor k_G, z_1, ..., z_{n_z}, p_1, ..., p_{n_p} \right\rfloor \tag{7}$$

where $n_z = n_{rz} + n_{cz}$ and $n_p = n_{rp} + n_{cp}$.

In case of a complex zero or pole, only the absolute value of the real and imaginary part is considered. Real coding avoids coding-decoding functions, and it is a more natural approach to a numerical optimization of engineering problems.

1.3.3.2. Crossover

Two different crossover operators have been used, switching between one and another randomly. In particular, the discrete crossover (exchanging parameters between individuals) and the intermediate crossover (weighting parameters). A good reference of them may be found in Michalewicz [16]. The definition of both crossover operators follows this paragraph.

Let two individuals $v_1 = (x_1^{v1}, ..., x_n^{v1})$ and $v_2 = (x_1^{v2}, ..., x_n^{v2})$, the crossover offspring ($v_{12} = cross(v_1, v_2)$) may be obtained by:

• Discrete crossover: $v_{12} = (x_1^{vi}, ..., x_n^{vi})$, where $i = \{1,2\}$ with the same probability.

• Intermediate crossover: $v_{21} = (a\, x_1^{v1} + (1- a\,)\, x_1^{v2}, ... , a\, x_n^{v1} + (1- a\,)\, x_n^{v2})$ where a is a random number between 0 and 1.

1.3.3.3. Mutation

This is the second evolutionary operator to create new individuals. For simplicity a uniform distribution (a bit uncommon in ES) is defined, which varies one of the parameters of the individual or with some probability all of them. Therefore, the mutation in parameter x_i of vector v may be expressed as:

$$mut(x_i) = x_i + \alpha(a - 0.5)(x_{i,\max} - x_{i,\min}) \tag{8}$$

where α is a real number, so that $0 \leq \alpha \leq 1$.

1.3.3.4. Selection

One of the more complex issues in evolutionary algorithms is the selection schema of individuals. In this sense, a deterministic and elitist approach is applied, where population is ordered according to a static fitness evaluation. In this way, a population of μ individuals, which

generates λ offspring, is reduced to the initial size selecting the μ individuals with the minimum fitness function from the total μ + λ individuals.

1.3.3.5. Algorithm Procedure

The work of the algorithm may be summarized in the following steps.

Step 1: Create an initial random population (ψ_0), where parameters are selected into their possible ranges.

Step 2: Select a number of individuals for crossover (ψ_{cross}) from the current population (ψ_t). If the number is not even, one of the individuals is added again. The crossover is performed successively every two individuals creating a new population as a result (ψ_{cross_new}). A pair of individuals produces two offspring. Considering groups of four individuals, the two best ones are selected to survive in ψ_{cross_new}.

Step 3: A number of individuals are selected from the current population to perform the mutation. For every individual the offspring is included in the resulting population (ψ_{mut}) if it is better than the original one. The non-mutated individuals are included into ψ_{no_mut}.

Step 4: A final population is built as the union of ψ_{cross_new}, ψ_{no_mut} and ψ_{mut}. The size is reduced to the initial one selecting the best individuals. They are the population for the next generation (ψ_{t+1}).

Step 5: If the maximum number of generations has not been reached, jump to step 2.

3.2.4. Fitness Evaluation

As it was mentioned previously, individuals are compared according to the evaluation of a fitness function, where an individual is better than any other if it causes a lower value in the fitness evaluation. This function is defined joining in an expression objectives and constraints. The coefficient election is very critical for the success in the search process. Definition of objectives and constraints to design a QFT controller is a subject with some degrees of freedom. It has been solved in the following manner.

1.3.4.1. Objective Functions

The set of objective functions must express the controller quality and the desired behaviour of it on the NC. To simplify the mathematical expressions in the rest of this strategy, the following notation is introduced:

$$\delta(condition) = \begin{cases} 1 & iff \quad condition\ true \\ 0 & iff \quad condition\ false \end{cases} \tag{9}$$

The set of work frequencies is *I*, and sub-index *i* always refers to a frequency of this set, $i \in I$. Also, the phase of $L_0(s)$ for an individual *x* is expressed as:

$$\theta_{x,i} = \angle L_0(x, \omega_i) \tag{10}$$

To improve quality a minimization of gain k_G is required. So, the first objective function $f_1(x)$ can be defined as:

$$f_1(x) = k_G^2 \tag{11}$$

Two other objective functions are included to manage the controller behaviour on the NC. The function $f_2(x)$ express an heuristic objective, looking for controllers with a $L_0(s)$ plot as near as possible to the UHFB (*Universal High Frequency Boundary*) bound, if frequency is greater than a reference ω_{UHFB} ($\omega_i > \omega_{UHFB}$). This reference is selected among the set of frequencies of the QFT analysis where bounds start to be in convergence with the UHFB. Also, the phase distance is only considered if it is bigger than the reference ϕ_{ref}, i.e., ($|\theta_{UHFB} - \theta_{x,i}| > \phi_{ref}$), where ϕ_{ref} is defined by the designer. See the next Fig.3 to clarify the content of this paragraph.

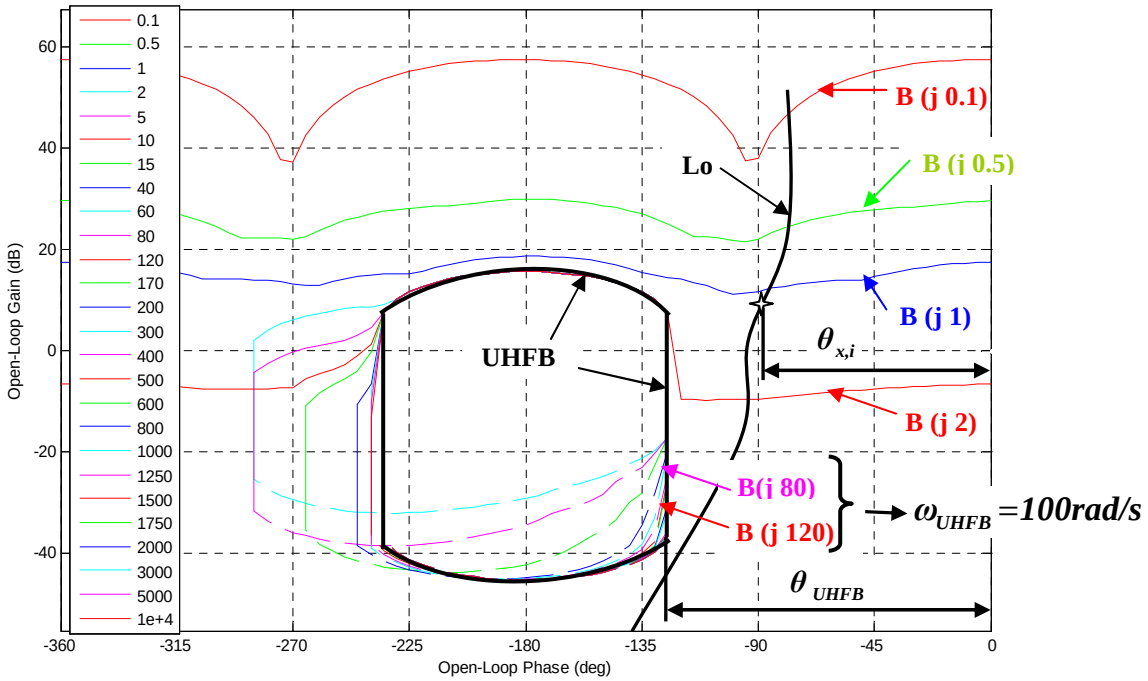

Fig.3. Clarification for the $f_2(x)$ objective function.

$$f_2(x) = \sum_{i \in I} \delta(\omega_i > \omega_{UHFB}) \delta\left(|\theta_{UHFB} - \theta_{x,i}| > \phi_{ref}\right) |\theta_{UHFB} - \theta_{x,i}| \tag{12}$$

There are situations where it is very useful to include as an objective the search of one controller achieving a descending phase for $L_0(s)$. This can be set by:

$$f_3(x) = \sum_{i \in I} \delta\left(\theta_{x,i+1} > \theta_{x,i}\right) \left| \theta_{x,i+1} - \theta_{x,i} \right| \tag{13}$$

These last two functions [$f_2(x)$, $f_3(x)$] might not be convenient in some situations. Designer should check the bounds on the NC to decide whether they are suitable. They are included here because of the usefulness to treat some specific problems.

1.3.4.2. Constraint Functions

Constraints are expressed by functions $g_j(x)$. They are useful to know whether an individual meets a condition, and if not, to measure the degree of mismatch. When an individual meets all restrictions it is called a feasible individual. Otherwise it is an unfeasible solution of the problem. In the next paragraphs the functions that have been considered for this study are introduced. When a controller meets the QFT bounds (over upper bounds q_u, or under lower bounds q_l, see Fig.4), one of the following conditions is true:

$$\begin{aligned} q_u(\theta_{x,i}, \omega_i) &\le L_0(x, \omega_i) \\ L_0(x, \omega_i) &\le q_l(\theta_{x,i}, \omega_i) \end{aligned} \quad i \in I \tag{14}$$

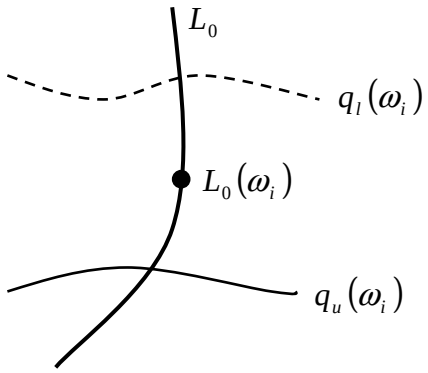

Fig.4. Clarification for Equation (14).

To check this condition, and quantify the level of bound violation if it is necessary, the following constraint functions are taken:

$$V(x) = \sum_{i \in I} V_\omega^2(x, \omega_i) \tag{15}$$

and,

$$g_1(x) = \delta(V(x) > 0)(K_V V(x) + 1) \tag{16}$$

where, $K_V = 1/r_1$ with r_1 the weight of the $g_1(x)$ constraint function, and where the sub-functions $V_\omega(x,\omega_i)$ explain the degree of mismatch of every bound at frequency ω_i. When an individual meets their correspondent bounds, the value is 0. Otherwise the value will be the minimum between the distance to the upper and to the lower bound. In a more formal way it can be written as:

$$V_u(x,\omega_i) = \max\{\log q_u(\theta_{x,i},\omega_i) - \log L_0(x,\omega_i), 0\}$$
$$V_l(x,\omega_i) = \max\{\log L_0(x,\omega_i) - \log q_l(\theta_{x,i},\omega_i), 0\} \quad (17)$$

and finally,

$$V_\omega(x,\omega_i) = \begin{cases} \min\{V_u(x,\omega_i), V_l(x,\omega_i)\} \ \textit{iff} \ \ q_u(\theta_{x,i},\omega_i) \geq q_l(\theta_{x,i},\omega_i) \\ \max\{V_u(x,\omega_i), V_l(x,\omega_i)\} \ \textit{iff} \ \ q_u(\theta_{x,i},\omega_i) < q_l(\theta_{x,i},\omega_i) \end{cases} \quad (18)$$

To check stability, roots of the characteristic equation [1+ $L_0(s)$] are computed. The real part of the root (d_i) is the distance to the complex axis. Violation of the stability constraint appears if the root is in the RHP (*Right-Half Plane*). Therefore, a second constraint function is defined:

$$d_T = \sum_{i=1}^{n} \delta(d_i > 0) d_i \quad (19)$$

and,

$$g_2(x) = \delta(d_T > 0)(K_{st}\, d_T + 1) \quad (20)$$

where, $K_{st} = 1/r_2$ with r_2 the weight of the $g_2(x)$ constraint function.

Finally, sometimes controllers that achieve a descending module plot for $|L_0(s)|$ are strongly preferred. Every controller without this feature is considered unfeasible. This avoids controllers with resonant frequencies. The constraint function can be defined as:

$$g_3(x) = \delta(|L_0(s)| \textit{ always descending}) \quad (21)$$

As it may be observed, the optimization procedure should reach feasible solutions, that means $g_1(x) = g_2(x) = g_3(x) = 0$, trying to minimize the objective functions $f_1(x)$, $f_2(x)$ and $f_3(x)$. These functions are non-linear and present a rather complex behaviour.

3.2.5. Application and Results

A QFT benchmark example is proposed to test the quality of this methodology. This benchmark is the same as the one selected by Chen et al. [8], Chait et al. [6], and Borghesani et al. [3] in previous works. The considered plant has a parameter uncertainty given by:

$$P=\left\{P(s)=\frac{k\,a}{s\,(s+a)}\mid k\in[1,10],a\in[1,10]\right\} \tag{22}$$

The nominal plant $P_0(s)$ is expressed so that the parameters are a=1 and k=1. The specifications for robust stability and reference tracking are respectively:

$$\left|\frac{P(s)G(s)}{1+P(s)G(s)}\right|\leq 1.2,\qquad \forall P\in \mathrm{P},\omega\geq 0 \tag{23}$$

for the stability, and,

$$T_L(s)\leq\left|\;F(s)\frac{P(s)G(s)}{1+P(s)G(s)}\right|\leq T_U(s) \tag{24}$$

for reference tracking, where,

$$T_U(s)=\left|\;\frac{0.6854(s+30)}{s^2+4s+19.752}\right| \tag{25}$$

and

$$T_L(s)=\left|\;\frac{120}{s^3+17s^2+82s+120}\right| \tag{26}$$

for $\omega>10$ rad/s.

Solutions given by Chen et al. [8] and Borghesani et al. [3] are:

$$G_{Chen}(s)=\frac{1.794s+6.203}{0.000004s^2+0.00274s+0.826} \tag{27}$$

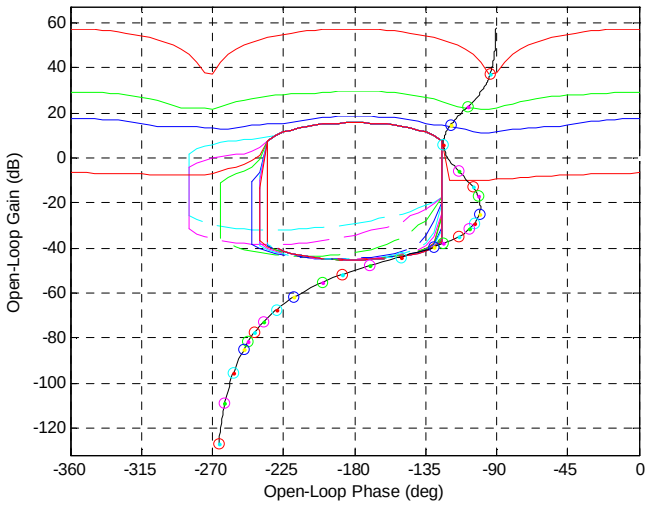

Fig.5. Nichols plot for $L_o(s)$ with $G_{Chen}(s)$

$$G_{Borghesani}(s)=\frac{3.0787\times10^{6}\,s^{2}+3.5365\times10^{8}\,s+3.8529\times10^{8}}{s^{3}+1.5288\times10^{3}\,s^{2}+1.0636\times10^{6}\,s+4.281\times10^{7}} \tag{28}$$

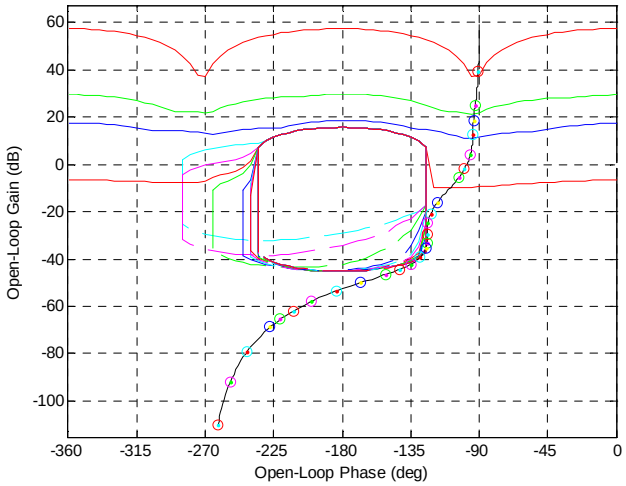

Fig.6. Nichols plot for $L_o(s)$ with $G_{Borghesani}(s)$

As can be seen, Chen et al. [8] provided a second order controller, whereas Borghesani et al. [3] provided a third order one. Their characteristics are summarized in the next table:

	$L_o(s)$ (Chen)	***$L_o(s)$ (Borghesani)***
Phase margin	59.15º	82º
Gain margin	49.92	52.66 dB
HFG	4.4852×10^{5}	3.079×10^{6}

Table 1: Chen and Borghesani controllers

Note: the HFGs (High Frequency Gain) have been computed according to the formula in [6]:

$$HFG=\lim_{s\to\infty}s^{r}\,G(s) \tag{29}$$

where r is the excess of poles of $G(s)$.

The high-frequency gain (HFG) of 4.4852×10^{5} in Chen is smaller than that of Borghesani of 3.079×10^{6}**.** This means that the Chen´s controller needs less control effort and is less sensitive to high frequency noise. In addition, while the gain margins are quite similar, the phase margin given by Chen (59.15º) is quite smaller than the one given by Borghesani (82º).

Now, the methodology developed in this work is applied to the problem in order to find controllers and to compare the results. To begin with, the same controller structure of Borghesani´s is imposed: a gain, two real zeros, a real pole and a pair of conjugate complex poles. Individual is expressed by the search vector [k_G, z_1, z_2, $p_{1,real}$, $p_{1,imag}$, p_2]. Wide ranges are defined for the parameters: $k_G \in [0.01,1500]$, $z_{1,2} \in [0.01,400]$, $p_{1,real} \in [0.01,1500]$, $p_{1,imag} \in [0.01,1500]$ and $p_2 \in [0.01,1500]$. Throughout the execution, the controller structure changes applying the

optimal Hankel norm approximation [37] to obtain a model reduction. It is important to emphasize that no initial controller to start the search is needed. Only an initial structure and ranges of variation for the parameters are needed. That is one of the strongest features of the evolutionary algorithms.

The complete set of frequencies to loop shape the controller is $\omega = \{0.1, 0.5, 1, 2, 5, 10, 15, 40, 60, 80, 120, 170, 200, 300, 400, 500, 600, 800, 1000, 1250, 1500, 1750, 2000, 3000, 5000, 10000\}$ (rad/s). Crossover probability is 0.25, mutation probability of a parameter is 0.08 and $\alpha = 0.25$. The number of generations is 300 and the population size is 80 individuals in each iteration. Due to the sensitivity of the evolutionary strategies to the fitness function coefficients, some of them have been considered fixed and others have been modified for different simulations. Coefficients K_v and K_{st} are 0.25 (=1/$r_{1,2}$), $\omega_{UHFB} = 100$ rad/s and $\phi_{ref} = 5^o$. Taking this into account, the methodology achieves the following solutions:

Simulation 1:

The coefficients for the fitness evaluation are $a_1 = 0.001$, $a_2 = 0.001$, $a_3 = 1$, $r_1 = [4,16]$, $r_2 = [4,16]$ and $r_3 = [4,16]$.

The unique feasible solution is a controller with the same structure as the one given by Borghesani et al. [3]:

$$G_1(s) = \frac{0.03009\,s^2 + 3.935s + 8.706}{5.272\times10^{-8}\,s^3 + 2.03\times10^{-5}\,s^2 + 0.02685\,s + 1} \tag{30}$$

It is quite similar to the one proposed by Borghesani et al. [3]. The Nichols plot of this controller may be observed in Figure 7.

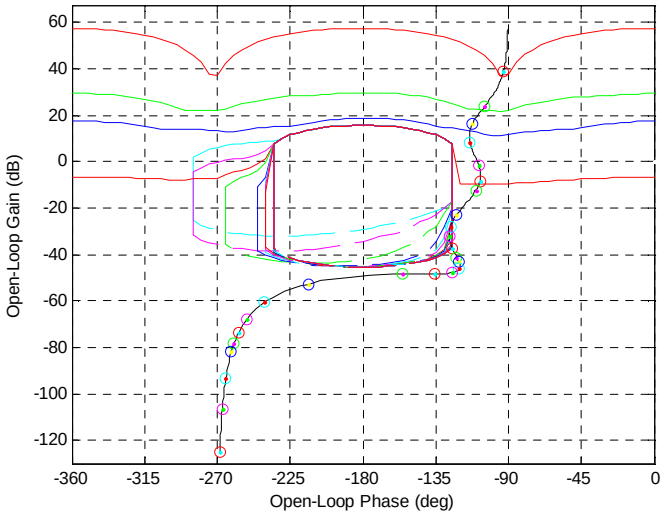

Fig.7. Nichols plot for $L_o(s)$ with $G_1(s)$

According to this, the controllers meet the problem requirements. The loop responses in frequencies corresponding to single-valued bounds are above their respective bound. Upper frequencies are out of the UHFB with a tangent response.

The next table shows the some performance properties of this controller:

	$L_o(s)$ (G_1)
Phase margin	79.69º
Gain margin	48.949 dB
HFG	5.7076×10^5

Table 2: $G_1(s)$ controller performance properties

Simulation 2:

The coefficients for the fitness evaluation are $a_1=0.001$, $a_2=0.001$, $a_3=2$, $r_1=[7,25]$, $r_2=[4,16]$ and $r_3=[4,16]$.

The controllers obtained in this simulation make $L_o(s)$ fulfil the bounds at every frequency defined for the bounds, but at frequencies between 2 and 5 rad/s the $L_o(s)$ crosses the UHFB, what is not desirable. This does not assure the closed loop stability at those frequencies for the plant with its uncertainty. The solutions are interesting even though they are not feasible.

The controllers are:

$$G_2(s)=\frac{0.01051s^2+3.092s+14.22}{7.062\times10^{-9}s^3+1.075\times10^{-5}s^2+0.005737s+1} \tag{31}$$

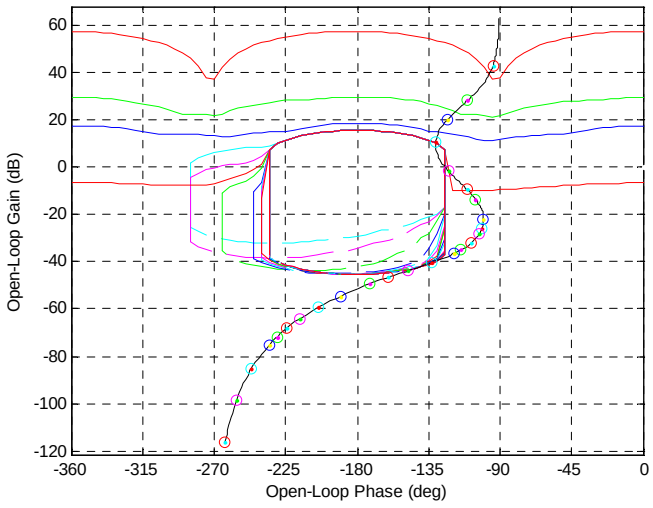

Fig.8. Nichols plot for $L_o(s)$ with $G_2(s)$

$$G_3(s) = \frac{3.048s + 14.22}{2.049\times10^{-6}s^2 + 0.002413s + 1} \tag{32}$$

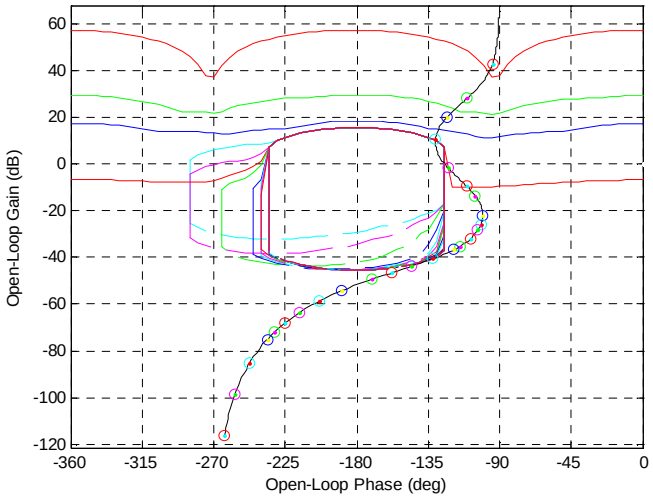

Fig.9. Nichols plot for $L_o(s)$ with $G_3(s)$

Fig.8 and Fig.9 show that the same loop-shape is obtained with both controllers, where $G_3(s)$ is the reduced model of $G_2(s)$ after applying the optimal Hankel norm approximation [32].

	$L_o(s)$ (G_2)	$L_o(s)$ (G_3)
Phase margin	55.3º	55.4º
Gain margin	51.4 dB	51.7 dB
HFG	1.4875×10^6	1.4874×10^6

Table 3: $G_2(s)$ and $G_3(s)$ controller's performance properties

Simulation 3:

The coefficients for the fitness evaluation are $a_1 = 0.001$, $a_2 = 0.001$, $a_3 = 2$, $r_1 = [6, 20]$, $r_2 = [4, 16]$ and $r_3 = [4, 16]$.

The controller obtained in this simulation is:

$$G_4(s) = \frac{0.01151s^2 + 3.454s + 7.522}{2.403\times10^{-8}s^3 + 9.379\times10^{-6}s^2 + 0.01293s + 1} \tag{33}$$

As in the previous simulation, here the closed loop stability is not assured at every frequency for the whole plant uncertainty range. Specifically, the stability is not assured between 120 and 170 rad/s for the worst case within the uncertainty. Apart from that, all the bounds are fulfilled at the corresponding frequencies.

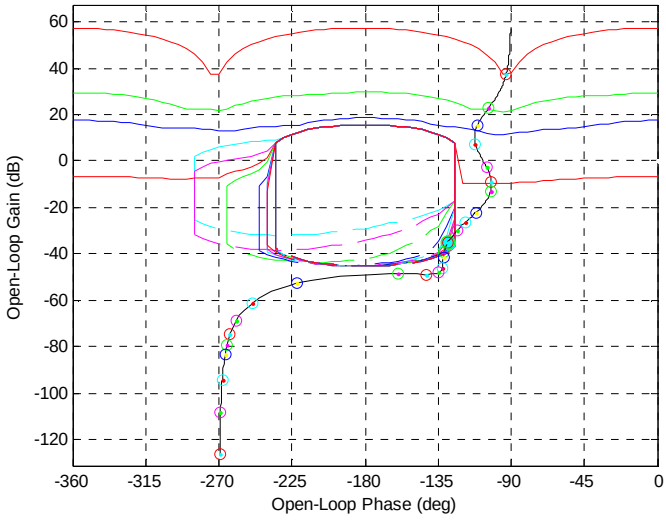

Fig.10. Nichols plot for $L_o(s)$ with $G_4(s)$

	$L_o(s)$ (G_4)
Phase margin	72.7º
Gain margin	48.9 dB
HFG	4.79×10^{5}

Table 4: $G_4(s)$ controller performance properties

Simulation 4:

The coefficients for the fitness evaluation are $a_1=0.001$, $a_2=0.001$, $a_3=2$, $r_1=[5,18]$, $r_2=[4,16]$ and $r_3=[4,16]$.

The feasible solution obtained in this simulation is:

$$G_5(s)=\frac{0.02007s^2+6.004s+8.499}{9.772\times10^{-9}s^3+5.556\times10^{-6}s^2+0.01173s+1} \tag{34}$$

Fig.11 shows that with this controller all the performance specification bounds are fulfilled, and that the closed loop stability is assured at every frequency.

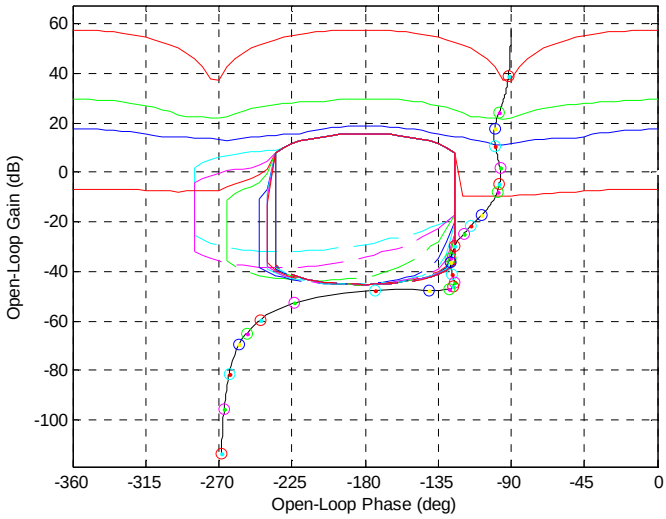

Fig.11. Nichols plot for $L_o(s)$ with $G_5(s)$

	$L_o(s)$ (G_5)
Phase margin	83.3º
Gain margin	47.7 dB
HFG	2.054×10^6

Table 5: $G_5(s)$ controller performance properties

	$L_o(s)$ (G_{Chen})	$L_o(s)$ ($G_{Borghesani}$)	$L_o(s)$ (G_1)	$L_o(s)$ (G_2)	$L_o(s)$ (G_3)	$L_o(s)$ (G_4)	$L_o(s)$ (G_5)
Phase margin	59.15º	82º	79.69º	55.3º	55.4º	72.7º	83.3º
Gain margin	49.92	52.66 dB	48.949 dB	51.4 dB	51.7 dB	48.9 dB	47.7 dB
HFG	4.4852×10^5	3.079×10^6	5.7076×10^5	1.487×10^6	1.487×10^6	4.79×10^5	2.054×10^6

Table 6: All the controllers

The feasible solutions found with this strategy are in the shadowed columns. Comparing these feasible solutions (G_1 and G_5) with the Chen´s and Borghesani´s, it is clear that G_1 and G_5 are quite similar to G_{Chen} and $G_{Borghesani}$.

The G_1 controller is quite similar to $G_{Borghesani}$ in terms of gain and phase margin, but G_1 has a smaller HFG. This means that G_1 needs less control effort and is less sensitive to high frequency noise. Comparing G_1 with G_{Chen} we see that while the gain margins are similar and the HFG are of the same order, the phase margin is lower in the case of G_{Chen}.

The G_5 controller is almost equal to $G_{Borghesani}$ because the values vary a little. Comparing it with G_{Chen}, their gain margins are similar, but the phase margin, as well as the HFG, is greater in the case of G_5 .

4. AUTOMATIC LOOP-SHAPING OF QFT ROBUST CONTROLLERS VIA GENETIC ALGORITHMS

4.1. INTRODUCTION

In this section the methodology a) applies Genetic Algorithms to search the QFT controller that fulfils the control specifications for the whole set of models with uncertainty, and b) validates the new approach by using several benchmarks.

The corresponding Matlab, C++ and Text files are described in Appendix B.

4.2. STRATEGY 2: GENETIC ALGORITHMS

4.2.1. Problem Statement

In this strategy the controller structure is expressed by the following transfer function:

$$G(x, j\omega) = k_G \frac{\prod_{i=1}^{n_z} (j\omega + z_i)}{\prod_{h=1}^{n_p} (j\omega + p_h)} \tag{35}$$

where, $x = \lfloor k_G, z_1, ..., z_{n_z}, p_1, ..., p_{n_p} \rfloor$ represents the parameters in the search space, being k_G the gain, z_i the real zeros (i=1,…,n_z) and p_h the real poles (h=1,…,n_p).

4.2.2. Genetic Algorithms

GAs (Genetic Algorithms) are search procedures based on the mechanics of natural selection and natural genetics. A population of candidate solutions (points in the search space) is maintained at every stage or every generation, using GAs nomenclature. For the next generation, a new set of artificial creatures is produced guided by the survival of the fittest principle. The idea is to improve performance by sampling areas of the search space that are more likely to lead to good solutions. Some features of GAs are [23]:

i. The parameter set x is coded as a finite-length string over some finite alphabet Γ.

ii. GAs search from a population of such strings, reducing the probability of finding local peaks. Furthermore, no initial point is used to start from.

iii. GAs use only the information provided by an objective (or fitness) function. No other auxiliary information is required.

iv. GAs use probabilistic transition rules to guide the search.

A simple GA is composed of three operators. First, reproduction, by which individual strings are copied to the next generation. Strings with a higher fitness value have a higher probability of contributing one or more offspring. Then, crossover, by which some newly created members are mated at random. From two mated strings, two new strings are created by swapping some characters between them. Finally, mutation is the random alteration (with very small probability) of the value of a character in the string.

The mathematical background behind GAs is the concept of similarity template or schema. A similarity template is a string from the extended alphabet $\Gamma^* = \Gamma \cup \{*\}$ where * is the do not care symbol. It turns out that the number of such templates implicitly processed in each generation is about n^3, where n is the number of members in the population (see [23] and [25]). This implicit parallelism gives the GA much of its power.

4.2.3. Proposed Algorithm

The software developed for the GA-based automatic loop-shaping is a C++ adaptation of the Simple Genetic Algorithm in [23]. The design decisions made in the program are the following:

i. The inverse of the cost function (37) is used – GAs are usually designed to maximise, rather than minimise.

ii. Fitness scaling, as proposed by Goldberg [23], is applied to obtain the actual objective function.

iii. The selected alphabet is $\Gamma = \{0, 1\}$, that is to say, binary strings code the parameters to optimise.

iv. The *i*-th parameter in the search space is coded with l_i bits.

v. The final user is requested to provide range and precision of the parameters by entering values for $x_{i,min}$, $x_{i,max}$ – minimum and maximum allowed values for the parameter x_i – and for the precision π_i, defined as,

$$\pi_i = \frac{x_{i,\max} - x_{i,\min}}{2^{l_i} - 1}; \quad i = 1,...,r; \quad r = 1 + n_z + n_p \tag{36}$$

where *r* is the number of parameters, a value the user can also modify. Equation (36) is used by the program to solve for l_i.

vi. The multi-parameter coding used in the program is the result of concatenating *r* single-parameter codings. Every candidate string is decoded to *r* unsigned integer values in the ranges $[0, 2^{li-1}]$, and then these values are mapped linearly to the ranges $[x_{i,min}, x_{i,max}]$.

vii. The initial population is randomly generated.

viii. The reproduction operator is based on the stochastic remainder selection without replacement (see [4] and [23]), where the expected number of offspring for each string is calculated, the integer part of these values directly allocated and the fractional parts treated as probabilities to complete the population.

ix. Finally, the user must provide the following additional parameters to run the program: crossover probability, mutation probability, number *n* of strings in every population, and the cost parameters $\omega_1, \omega_2, a_0, a_1, a_2$, and a_3.

4.2.4. Fitness Evaluation

Following the cost function proposed by Thompson and Nwokah [38] and Thompson [39], a new reformulation is introduced such that,

$$J(x)=a_0 k_G^2 + a_1 \sum_{i=1}^{n_z} \frac{(p_i - z_i)^4 + 1}{p_i z_i} + a_2 A(\omega_1, \omega_2) + a_3 V(x) \tag{37}$$

where

$$A(\omega_1, \omega_2)= \int_{\omega_1}^{\omega_2} \ln\left|G(j\omega)\right| d\omega \tag{38}$$

and where *V(x)* is a penalty function, to be discussed below.

This new cost function is based on Horowitz's original notion of optimum, which is related with the well-known cost of feedback [26]. The area term allows tighter control of the gain over any frequency range of interest. Indeed, it will be used to reduce over-design at low frequencies. Moreover, additional area terms for other frequency ranges could be added when necessary. The *V(x)* term in (37) penalises unfeasible solutions, i.e. those that do not satisfy the stability and performance specifications.

As mentioned above, the key of QFT is a transformation of robust stability and robust performance specifications and plant uncertainties into domains in the complex plane, referred to as bounds, where a nominal loop transmission should lie within [7]. A generalised bound can be represented as a function *q* of phase and frequency, composed of upper and lower parts, q_u and q_l respectively, such that

$$\begin{aligned} Lm\, q_u(\angle L_0(x, j\omega_i), \omega_i) &\le Lm\, L_0(x, j\omega_i) \\ Lm\, L_0(x, j\omega_i) &\le Lm\, q_l(\angle L_0(x, j\omega_i), \omega_i) \end{aligned} \qquad i \in I \tag{39}$$

where $L_0=GP_0$ and Lm denotes $\log_{10}$ magnitude, *I* represents the discrete set of frequencies of interest and P_0 is the selected nominal plant. For convenience, it is denoted,

$$\theta(x,\omega_i)=\angle L_0(x,\, j\omega_i) \tag{40}$$

Let

$$\begin{aligned} V_u(x,\omega_i) &= \max\left(Lm\, q_u(\theta(x,\omega_i),\omega_i) - Lm\, L_0(x,\, j\omega_i), 0\right) \\ V_l(x,\omega_i) &= \max\left(Lm\, L_0(x,\, j\omega_i) - Lm\, q_l(\theta(x,\omega_i),\omega_i), 0\right) \end{aligned} \tag{41}$$

represent the degree of violation of an upper or lower bound, and define

$$V_\omega(x,\omega_i)=\begin{cases} \min(V_u(x,\omega_i), V_l(x,\omega_i)), \\ \textit{if } Lm\, q_u(\theta(x,\omega_i),\omega_i) \geq Lm\, q_l(\theta(x,\omega_i),\omega_i), \\ \max(V_u(x,\omega_i), V_l(x,\omega_i)), \\ \textit{otherwise.} \end{cases} \tag{42}$$

Then, the penalty function is defined as follows:

$$V(x)=\sum_{i\in I}\left(V_\omega(x,\omega_i)\right)^2 \tag{43}$$

which completes the cost function in (37).

The GA program simply read the numerically computed bounds from a text file, so any type of bound or combinations of bounds could be used. For convenience, the QFT Toolbox of MATLAB [3] is used to generate the bounds. Any problem that can be stated with that tool, or with any other, as long as a text file with the bounds can be generated, is potentially solvable with this methodology. Also, note that no especial requirements are imposed on the plant templates.

Typically, CAD tools compute bounds only for a discrete set of phases varying in a certain range, say [-2π, 0]. In order to evaluate the penalty term for phase values where q_l and q_u have not been computed, a linear interpolation was used. Of course, this is only a convenient, simple-to-implement solution. Otherwise, an analytical equation for the bounds could be directly programmed when available. In this respect, [7] showed that there exists a formal map from the uncertain plant and each closed loop specification to the bounds, which could be written as programmable quadratic inequalities.

4.2.5. Application and Results

Here another QFT benchmark example is proposed to test the quality of the new methodology. This example was taken by Thompson and Nwokah [38] from a flight control problem due to Blakelock [2]. The plant is:

$$P(s)=\frac{k\left(1+\frac{s}{z}\right)}{s\left(1+\frac{s}{p}\right)\left(1+\frac{2\xi}{\omega_n}s+\frac{s^2}{\omega_n^2}\right)} \tag{44}$$

where the uncertain parameters are given by,

$$k\in[0.2,2],\ z\in[0.5,75],\ p\in[1,10],\ \omega_n\in[5,6],\ \xi\in[0.8,0.9] \tag{45}$$

The nominal plant selected for the loop-shaping is

$$P_0(s)=\frac{k_0\left(1+\frac{s}{z_0}\right)}{s\left(1+\frac{s}{p_0}\right)\left(1+\frac{2\xi_0}{\omega_{n0}}s+\frac{s^2}{\omega_{n0}^2}\right)} \tag{46}$$

$$k_0=2,\ z_0=0.5,\ p_0=10,\ \omega_{n0}=6,\ \xi_0=0.8$$

Equations (47), (48), (49) and (50) exhibit the desired closed-loop specifications:

$$\left|\frac{P(s)G(s)}{1+P(s)G(s)}\right|\le 6\,dB,\quad \forall P\in \mathrm{P}\text{, and } \omega\in\{0.01,0.05,0.1,0.2,1,5,10,50,100\} \tag{47}$$

for the stability, and

$$T_L(s)\le\left|F(s)\frac{P(s)G(s)}{1+P(s)G(s)}\right|\le T_U(s) \tag{48}$$

for reference tracking, where,

$$T_U(s)=\left|\frac{1+\left(\frac{s}{0.35}\right)}{\left(1+\frac{s}{0.5}\right)\left(1+\frac{s}{3}\right)}\right| \tag{49}$$

and

$$T_L(s)=\left|\frac{1}{(1+s)(1+s)\left(1+\frac{s}{2}\right)}\right| \tag{50}$$

for $\omega\in\{0.01,0.05,0.1,0.2,1,5,10\}$.

Thompson and Nwokah [38] started from the H_∞ controller,

$$G_{H_\infty}(s)=\frac{7.709\left(1+\frac{s}{0.882}\right)\left(1+\frac{s}{2.22}\right)\left(1+\frac{s}{2.22}\right)\left(1+\frac{s}{7}\right)\left(1+\frac{s}{7}\right)}{\left(1+\frac{s}{0.51}\right)\left(1+\frac{s}{0.615}\right)\left(1+\frac{s}{2.6}\right)\left(1+\frac{s}{28.36}\right)\left(1+\frac{s}{1000}\right)} \tag{51}$$

to obtain, after a non-linear optimisation procedure and limiting each controller parameter to a $\pm$40 percent neighbourhood of its initial value,

$$G_{TN}(s)=\frac{371.003(s+0.630)(s+2.994)(s+2.994)(s+7.035)(s+7.035)}{(s+0.714)(s+0.822)(s+3.640)(s+20.254)(s+714.29)} \tag{52}$$

As required by their optimisation technique, the plant templates were approximated by the smallest possible super scribed rectangle. A 55% reduction in the high frequency gain was obtained.

Figure.12 and Table.7 show the loop-shaping and the performance properties respectively for the G_{TN} controller:

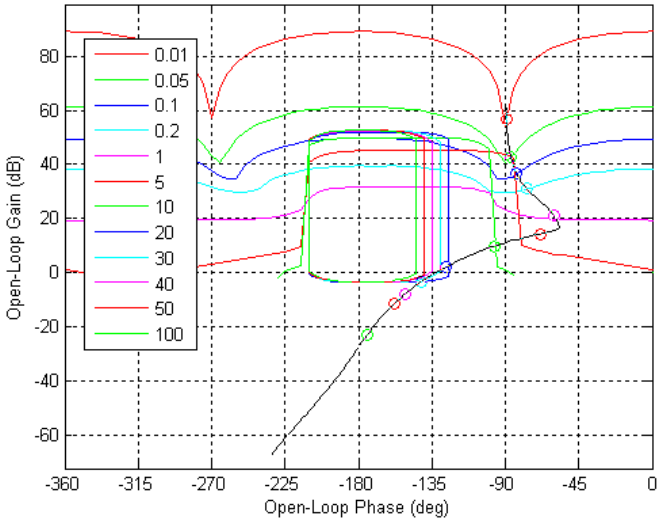

Fig.12. Nichols plot for $L_o(s)$ with $G_{TN}(s)$

	$L_o(s)$ (G_{TN})
Phase margin	47.2°
Gain margin	27.4 dB
HFG	3.71×10^2

Table 7: $G_{TN}(s)$ controller´s performance properties

Note: the HFGs (High Frequency Gain) are computed according to the formula in [6]:

$$HFG = \lim_{s \to \infty} s^r \, G(s) \tag{53}$$

where *r* is the excess of poles of *G(s).*

Now, the GA methodology developed in this work is applied to the problem in order to compare and validate the results. To begin with, the same controller structure of Thompson and Nwokah is imposed: a gain, five real zeros and five real poles. As the program still requires maximum and minimum parameter values, wide ranges are chosen: $k \in [0.01,1000]$, $z_{1,\ldots,3} \in [0.01,5]$, $z_{4,5} \in [0.01,10]$ and $p_{1,\ldots,5} \in [0.01,1000]$. With such wide ranges, the GA is given freedom enough to significantly reduce high frequency gain. To avoid low frequency over-design, the area term in the cost function, tuned in the low frequency range, is used. Cost parameters are set to $a_0 = 0.001$, $a_1 = 0.00001$, $a_2 = 1$, area of frequency range $[\omega_1, \omega_2] = [0.01,1]$, and $a_3 = 1$. Throughout the execution, the controller structure changes applying the optimal Hankel norm approximation [37] to obtain a model reduction.

The complete set of frequencies to loop shape the controller is ω = {0.01 0.05 0.1 0.2 1 5 10 20 30 40 50 100} (rad/s). Crossover probability is 0.1 and the mutation probability of a parameter is 0.01. The maximum number of generations is 500 and the population size is 100 individuals in each iteration. Due to the sensitivity of the genetic strategies to the fitness function coefficients, some of them have been considered fixed and others have been modified for different configurations.

It is important to emphasize that no initial controller to start the search is needed. Only an initial structure and ranges of variation for the parameters are needed (as it happened with the Evolutionary Strategy in section 3).

For comparison reasons with the Thompson and Nwokah [38] controller, the rectangular approximation for the templates has been applied, even though it is not necessary to run this methodology.

As it was mentioned in a previous paragraph, different configurations are chosen varying some parameters while others are kept fixed. The results obtained are the following:

Simulation 1:

0.01	a_0 parameter
0.0001	a_1 parameter
1	a_2 parameter
0.01	w_{Ini} to measure the "excess controller bandwidth"
1	w_{Fin} to measure the "excess controller bandwidth"
2	a_3 parameter

1440 gain of the QFT nominal plant
1 number of zeros of the QFT nominal plant
0.5 zero of the nominal plant ; for complex: (2,3) just one of the conjugates
3 number of poles of the QFT nominal plant (without counting the conjugates)
0 pole of the nominal plant; for complex: (2,3) just one of the conjugates
10 pole of the nominal plant; for complex: (2,3) just one of the conjugates
(4.8, 3.6) pole of the nominal plant; for complex: (2,3) just one of the conjugates
11 number of parameters
0.01 minimum value for the parameter 1
0.01 minimum value for the parameter 2
0.01 minimum value for the parameter 3
0.01 minimum value for the parameter 4
0.01 minimum value for the parameter 5
0.01 minimum value for the parameter 6
0.01 minimum value for the parameter 7
0.01 minimum value for the parameter 8
0.01 minimum value for the parameter 9
0.01 minimum value for the parameter 10
0.01 minimum value for the parameter 11
1000 maximum value for the parameter 1
5 maximum value for the parameter 2
5 maximum value for the parameter 3
5 maximum value for the parameter 4
10 maximum value for the parameter 5
10 maximum value for the parameter 6
1000 maximum value for the parameter 7
1000 maximum value for the parameter 8
1000 maximum value for the parameter 9
1000 maximum value for the parameter 10
1000 maximum value for the parameter 11
.01 desired precision for parameters
200 number of individuals
0.1 crossover probability
0.01 mutation probability
1 show continuously (1) or just at the end (0)
500 maximum number of generations
100000 minimum desired fitness

0 number of controller fixed poles
5 number of parameters that corresponds to controller zeros

The controllers obtained with this configuration are:

$$G_1(s)=\frac{126.711(s+3.311)(s+3.74)(s+4.053)(s+4.16)(s+5.811)}{(s+1.017)(s+1.063)(s+5.93)(s+23.19)(s+224.2)} \tag{54}$$

for the original structure (similar to G_{TN}).

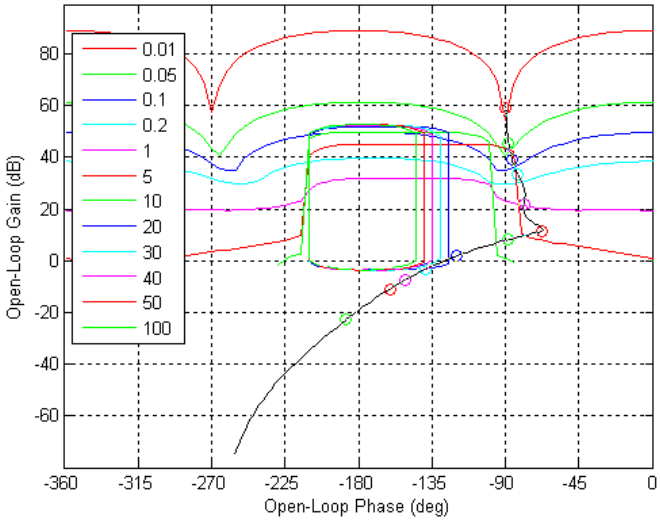

Fig.13. Nichols plot for $L_o(s)$ with $G_1(s)$

$$G_2(s)=\frac{126.711(s^2+6.476s+10.63)(s^2+8.672s+19.27)}{(s+224.2)(s+23.19)(s+1.07)(s+1.011)} \tag{55}$$

with a structure simplified once.

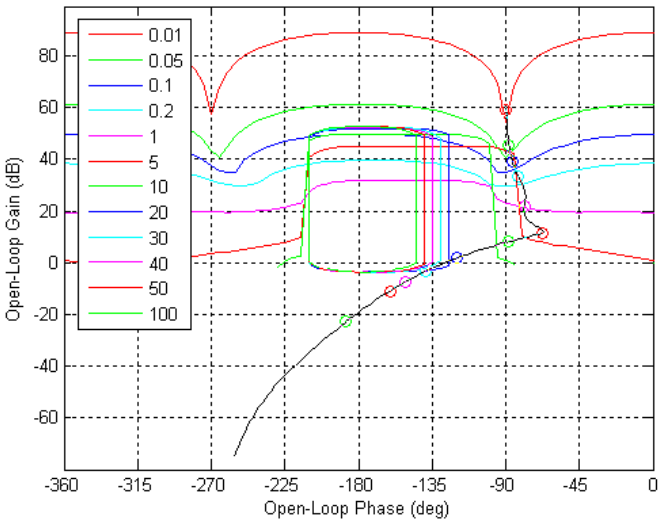

Fig.14. Nichols plot for $L_o(s)$ with $G_2(s)$

	$L_o(s)$ (G_1)	$L_o(s)$ (G_2)
Phase margin	52.9º	52.9º
Gain margin	19.2 dB	19.2 dB
HFG	1.26711×10^2	1.26711×10^2

Table 8: $G_1(s)$ and $G_2(s)$ controllers´ performance properties

Simulation 2:

0.01 a_0 parameter
0.0001 a_1 parameter
1 a_2 parameter
0.01 w_{Ini} to measure the "excess controller bandwidth"
1 w_{Fin} to measure the "excess controller bandwidth"
2 a_3 parameter
1440 gain of the QFT nominal plant
1 number of zeros of the QFT nominal plant
0.5 zero of the nominal plant ; for complex: (2,3) just one of the conjugates
3 number of poles of the QFT nominal plant (without counting the conjugates)
0 pole of the nominal plant; for complex: (2,3) just one of the conjugates
10 pole of the nominal plant; for complex: (2,3) just one of the conjugates
(4.8, 3.6) pole of the nominal plant; for complex: (2,3) just one of the conjugates

11 number of parameters
0.01 minimum value for the parameter 1
0.01 minimum value for the parameter 2
0.01 minimum value for the parameter 3
0.01 minimum value for the parameter 4
0.01 minimum value for the parameter 5
0.01 minimum value for the parameter 6
0.01 minimum value for the parameter 7
0.01 minimum value for the parameter 8
0.01 minimum value for the parameter 9
0.01 minimum value for the parameter 10
0.01 minimum value for the parameter 11
1000 maximum value for the parameter 1
5 maximum value for the parameter 2
5 maximum value for the parameter 3
5 maximum value for the parameter 4
10 maximum value for the parameter 5
10 maximum value for the parameter 6
1000 maximum value for the parameter 7
1000 maximum value for the parameter 8
1000 maximum value for the parameter 9

1000 maximum value for the parameter 10
1000 maximum value for the parameter 11
.01 desired precision for parameters
300 number of individuals
0.1 crossover probability
0.01 mutation probability
1 show continuously (1) or just at the end (0)
500 maximum number of generations
100000 minimum desired fitness

0 number of controller fixed poles
5 number of parameters that corresponds to controller zeros

The controllers obtained with this configuration are:

$$G_3(s)=\frac{105.662(s+4.063)(s+4.385)(s+4.385)(s+4.883)(s+9.092)}{(s+26.77)(s+11.42)(s+159.4)(s+1.566)(s+1.322)} \tag{56}$$

for the original structure (similar to G_{TN}).

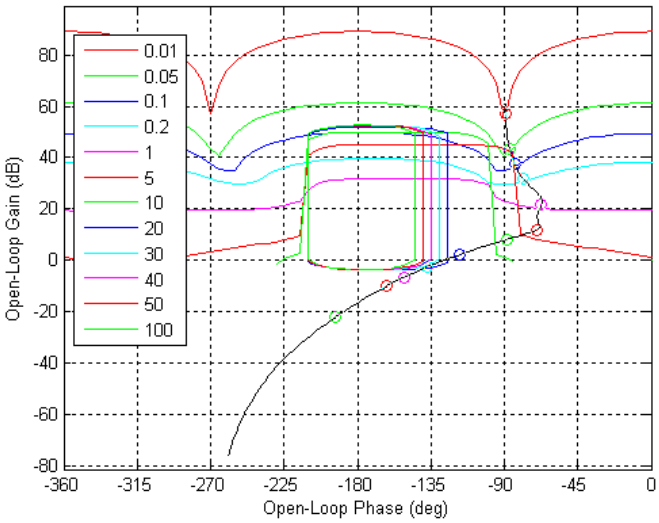

Fig.15. Nichols plot for $L_o(s)$ with $G_3(s)$

$$G_4(s)=\frac{105.662(s^2+5.677s+10.05)(s^2+10.67s+36.37)}{(s+157.4)(s+28.08)(s+1.974)(s+1.217)} \tag{57}$$

with a structure simplified once.

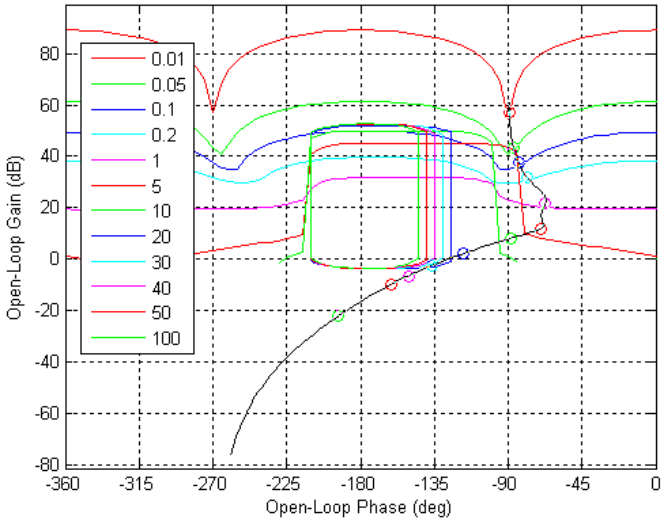

Fig.16. Nichols plot for $L_o(s)$ with $G_4(s)$

$$G_5(s) = \frac{63.0897(s+4.961)(s+4.365)(s+3.525)(s+8.164)}{(s+0.4494)(s+4.328)(s+43.46)(s+71.38)} \tag{58}$$

applying the GAs with the previous simplified structure.

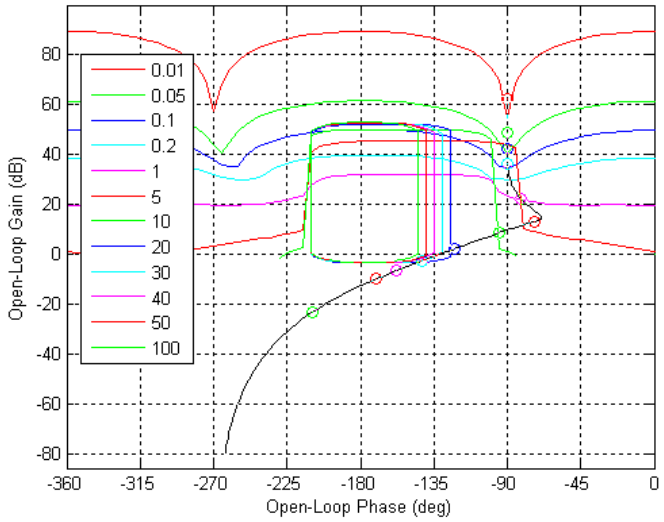

Fig.17. Nichols plot for $L_o(s)$ with $G_5(s)$

	$L_o(s)$ (G_3)	$L_o(s)$ (G_4)	$L_o(s)$ (G_5)
Phase margin	53.2º	53.4º	49.2º
Gain margin	16.6 dB	16.6 dB	12.8 dB
HFG	1.05662×10^2	1.05662×10^2	0.6309×10^2

Table 9: $G_3(s)$, $G_4(s)$ and $G_5(s)$ controllers´ performance properties

Simulation 3:

0.01	a_0 parameter
0.0001	a_1 parameter
1	a_2 parameter
0.01	w_{Ini} to measure the "excess controller bandwidth"
1	w_{Fin} to measure the "excess controller bandwidth"
3	a_3 parameter
1440	gain of the QFT nominal plant
1	number of zeros of the QFT nominal plant
0.5	zero of the nominal plant ; for complex: (2,3) just one of the conjugates
3	number of poles of the QFT nominal plant (without counting the conjugates)
0	pole of the nominal plant; for complex: (2,3) just one of the conjugates
10	pole of the nominal plant; for complex: (2,3) just one of the conjugates
(4.8, 3.6)	pole of the nominal plant; for complex: (2,3) just one of the conjugates

11	number of parameters
0.01	minimum value for the parameter 1
0.01	minimum value for the parameter 2
0.01	minimum value for the parameter 3
0.01	minimum value for the parameter 4
0.01	minimum value for the parameter 5
0.01	minimum value for the parameter 6
0.01	minimum value for the parameter 7
0.01	minimum value for the parameter 8
0.01	minimum value for the parameter 9
0.01	minimum value for the parameter 10
0.01	minimum value for the parameter 11
1000	maximum value for the parameter 1
5	maximum value for the parameter 2
5	maximum value for the parameter 3
5	maximum value for the parameter 4
10	maximum value for the parameter 5
10	maximum value for the parameter 6
1000	maximum value for the parameter 7
1000	maximum value for the parameter 8
1000	maximum value for the parameter 9
1000	maximum value for the parameter 10
1000	maximum value for the parameter 11
.01	desired precision for parameters
250	number of individuals
0.1	crossover probability
0.01	mutation probability
1	show continuously (1) or just at the end (0)
500	maximum number of generations

100000 minimum desired fitness
0 number of controller fixed poles
5 number of parameters that corresponds to controller zeros

The controllers obtained with this configuration are:

$$G_6(s) = \frac{282.968(s+4.023)(s+4.463)(s+4.902)(s+5.781)(s+6.787)}{(s+0.3533)(s+2.985)(s+7.327)(s+52.27)(s+249.7)} \tag{59}$$

for the original structure (similar to G_{TN}).

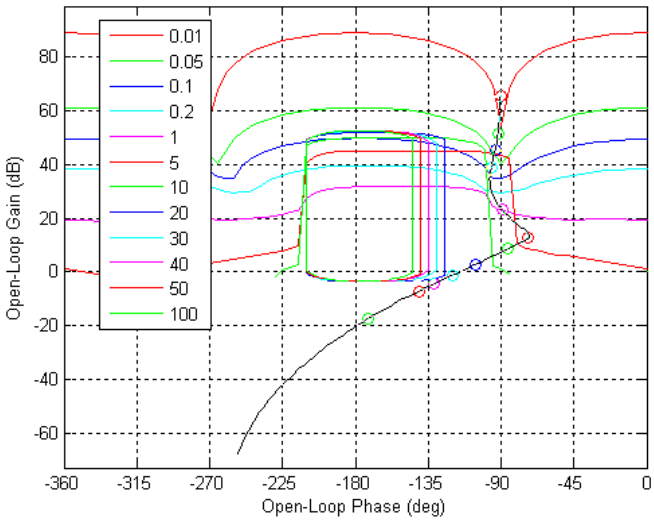

Fig.18. Nichols plot for $L_o(s)$ with $G_6(s)$

$$G_7(s) = \frac{282.968(s^2+7.61s+15.07)(s^2+11.15s+32.58)}{(s+249.6)(s+52.31)(s+3.107)(s+0.3533)} \tag{60}$$

with a structure simplified once.

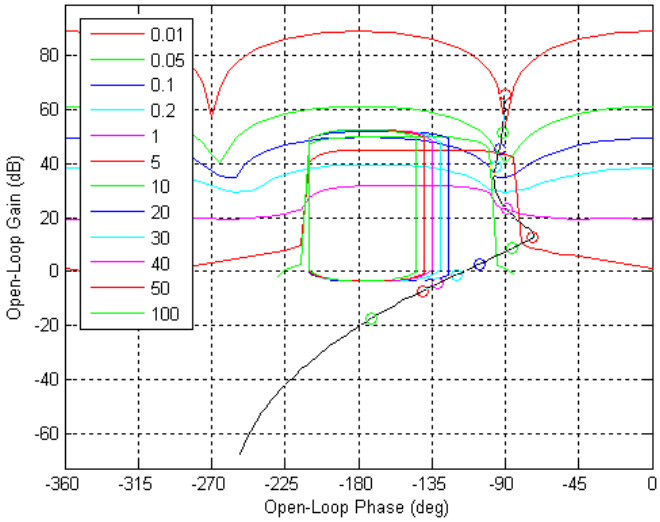

Fig.19. Nichols plot for $L_o(s)$ with $G_7(s)$

$$G_8(s) = \frac{48.777(s+3.477)(s+3.818)(s+3.916)(s+4.356)}{(s+1.104)(s+0.7806)(s+30.95)(s+70.96)} \tag{61}$$

applying the GAs with the previous simplified structure.

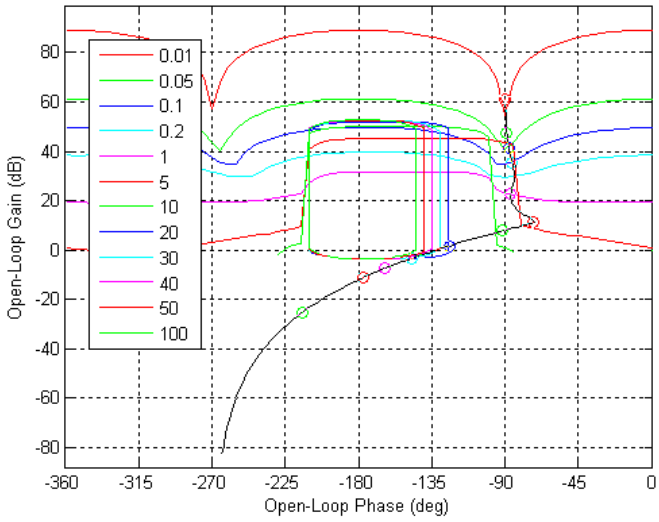

Fig.20. Nichols plot for $L_o(s)$ with $G_8(s)$

	$L_o(s)$ (G_6)	$L_o(s)$ (G_7)	$L_o(s)$ (G_8)
Phase margin	64.4º	64.4º	47.8º
Gain margin	20.5 dB	20.5 dB	12.1 dB
HFG	2.829×10^2	2.829×10^2	0.487×10^2

Table 10: $G_6(s)$, $G_7(s)$ and $G_8(s)$ controllers´ performance properties

Simulation 4:

0.01	a_0 parameter
0.0001	a_1 parameter
1	a_2 parameter
0.01	w_{Ini} to measure the "excess controller bandwidth"
1	w_{Fin} to measure the "excess controller bandwidth"
3	a_3 parameter
1440	gain of the QFT nominal plant
1	number of zeros of the QFT nominal plant
0.5	zero of the nominal plant ; for complex: (2,3) just one of the conjugates
3	number of poles of the QFT nominal plant (without counting the conjugates)
0	pole of the nominal plant; for complex: (2,3) just one of the conjugates
10	pole of the nominal plant; for complex: (2,3) just one of the conjugates
(4.8, 3.6)	pole of the nominal plant; for complex: (2,3) just one of the conjugates

11	number of parameters
0.01	minimum value for the parameter 1
0.01	minimum value for the parameter 2
0.01	minimum value for the parameter 3
0.01	minimum value for the parameter 4
0.01	minimum value for the parameter 5
0.01	minimum value for the parameter 6
0.01	minimum value for the parameter 7
0.01	minimum value for the parameter 8
0.01	minimum value for the parameter 9
0.01	minimum value for the parameter 10
0.01	minimum value for the parameter 11
1000	maximum value for the parameter 1
5	maximum value for the parameter 2
5	maximum value for the parameter 3
5	maximum value for the parameter 4
10	maximum value for the parameter 5
10	maximum value for the parameter 6
1000	maximum value for the parameter 7
1000	maximum value for the parameter 8
1000	maximum value for the parameter 9
1000	maximum value for the parameter 10
1000	maximum value for the parameter 11
.01	desired precision for parameters
300	number of individuals
0.2	crossover probability
0.01	mutation probability
1	show continuously (1) or just at the end (0)
500	maximum number of generations

100000 minimum desired fitness
0 number of controller fixed poles
5 number of parameters that corresponds to controller zeros

The controllers obtained with this configuration are:

$$G_9(s) = \frac{123.949(s+2.1)(s+4.063)(s+4.6)(s+9.805)}{(s+0.8791)(s+1.528)(s+81.18)(s+92.1)} \quad (62)$$

applying the GAs with a structure simplified once.

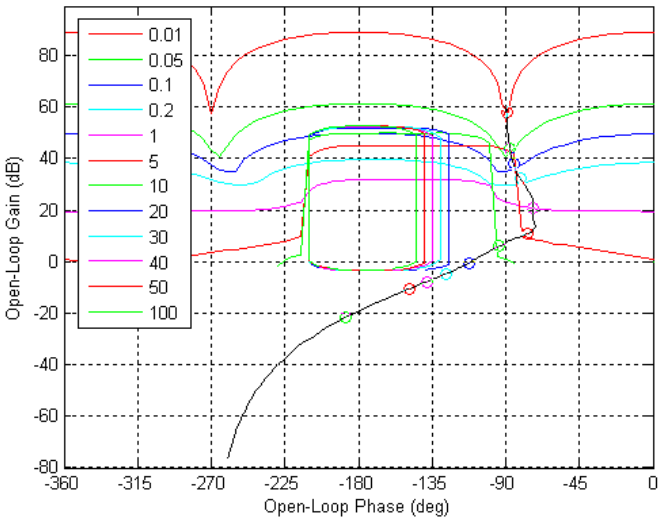

Fig.21. Nichols plot for $L_o(s)$ with $G_9(s)$

	$L_o(s)$ (G_9)
Phase margin	63.4º
Gain margin	17.4 dB
HFG	1.239×10^2

Table 11: $G_9(s)$ controller's performance properties

Simulation 5:

0.1 a_0 parameter
0.0001 a_1 parameter
1 a_2 parameter
0.01 w_{Ini} to measure the "excess controller bandwidth"
1 w_{Fin} to measure the "excess controller bandwidth"
3 a_3 parameter

1440 gain of the QFT nominal plant
1 number of zeros of the QFT nominal plant
0.5 zero of the nominal plant ; for complex: (2,3) just one of the conjugates
3 number of poles of the QFT nominal plant (without counting the conjugates)
0 pole of the nominal plant; for complex: (2,3) just one of the conjugates
10 pole of the nominal plant; for complex: (2,3) just one of the conjugates
(4.8, 3.6) pole of the nominal plant; for complex: (2,3) just one of the conjugates

11 number of parameters
0.01 minimum value for the parameter 1
0.01 minimum value for the parameter 2
0.01 minimum value for the parameter 3
0.01 minimum value for the parameter 4
0.01 minimum value for the parameter 5
0.01 minimum value for the parameter 6
0.01 minimum value for the parameter 7
0.01 minimum value for the parameter 8
0.01 minimum value for the parameter 9
0.01 minimum value for the parameter 10
0.01 minimum value for the parameter 11
1000 maximum value for the parameter 1
5 maximum value for the parameter 2
5 maximum value for the parameter 3
5 maximum value for the parameter 4
10 maximum value for the parameter 5
10 maximum value for the parameter 6
1000 maximum value for the parameter 7
1000 maximum value for the parameter 8
1000 maximum value for the parameter 9
1000 maximum value for the parameter 10
1000 maximum value for the parameter 11
.01 desired precision for parameters
250 number of individuals
0.15 crossover probability
0.01 mutation probability
1 show continuously (1) or just at the end (0)
500 maximum number of generations
100000 minimum desired fitness

0 number of controller fixed poles
5 number of parameters that corresponds to controller zeros

The controllers obtained with this configuration are:

$$G_{10}(s)=\frac{62.7769(s+2.705)(s+4.17)(s+4.707)(s+5.01)(s+8.691)}{(s+1.711)(s+0.5517)(s+7.639)(s+44.85)(s+69.92)} \tag{63}$$

for the original structure (similar to G_{TN}).

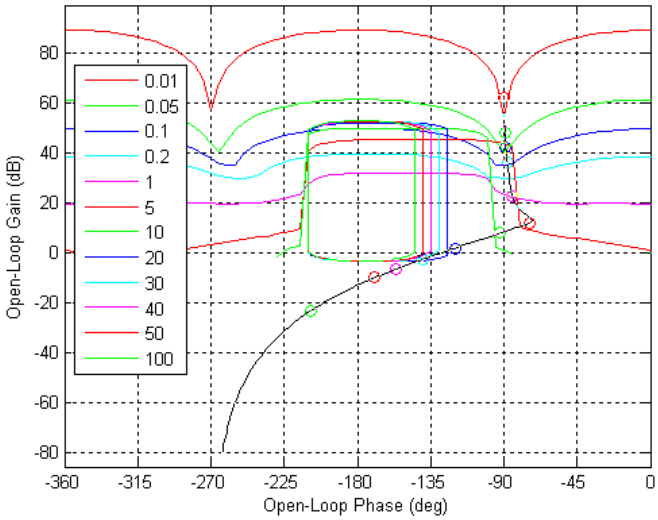

Fig.22. Nichols plot for $L_o(s)$ with $G_{10}(s)$

$$G_{11}(s)=\frac{62.7769(s+7.283)(s+2.391)(s^2+7.845s+16.79)}{(s+70.62)(s+44.24)(s+1.657)(s+0.5528)} \tag{64}$$

with a structure simplified once.

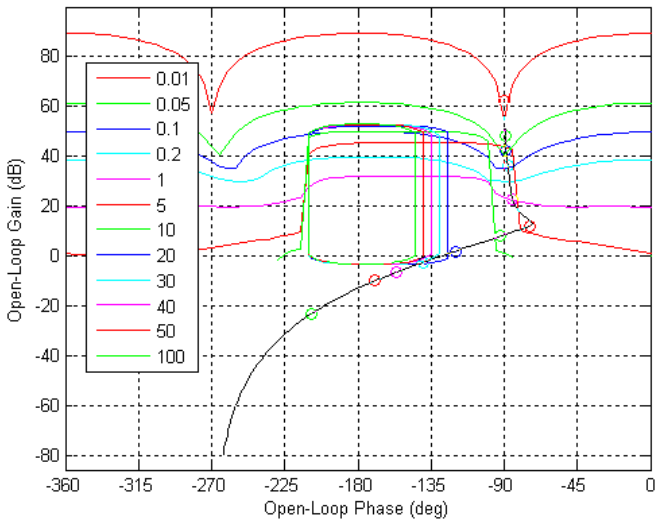

Fig.23. Nichols plot for $L_o(s)$ with $G_{11}(s)$

	$L_o(s)$ (G_{10})	$L_o(s)$ (G_{11})
Phase margin	52.4º	52.3º
Gain margin	13.2 dB	13.2 dB
HFG	0.627×10^2	0.627×10^2

Table 12: $G_{10}(s)$ and $G_{11}(s)$ controllers´ performance properties

	$L_o(s)$ (G_{TN})	$L_o(s)$ (G_1)	$L_o(s)$ (G_2)	$L_o(s)$ (G_3)	$L_o(s)$ (G_4)	$L_o(s)$ (G_5)	$L_o(s)$ (G_6)	$L_o(s)$ (G_7)	$L_o(s)$ (G_8)	$L_o(s)$ (G_9)	$L_o(s)$ (G_{10})	$L_o(s)$ (G_{11})
Phase margin	47.2º	52.9º	52.9º	53.2º	53.4º	49.2º	64.4º	64.4º	47.8º	63.4º	52.4º	52.3º
Gain margin	27.4 dB	19.2 dB	19.2 dB	16.6 dB	16.6 dB	12.8 dB	20.5 dB	20.5 dB	12.1 dB	17.4 dB	13.2 dB	13.2 dB
HFG	371	126.7	126.7	105.6	105.6	63.09	282.9	282.9	48.7	123.9	62.7	62.7
Number of poles + zeros	10	10	8	10	8	8	10	8	8	8	10	8

Table 13: All controllers´ performance properties

Table 13 compares the achieved controllers with the one given by Thompson and Nwokah [38]. As we can see, the phase margin of the new controllers is always higher (but close to 47.2º), the gain margin is always lower and the High Frequency Gain is in most cases quite lower than 371. In addition, the new controllers have a simpler (or the same) structure than the Thompson and Nwokah one. With these results it is possible to conclude that the new GA methodology applied in this work is good enough to achieve useful controllers as a first approach.

5. CONCLUSION

As it was concluded with the Evolutionary Algorithms, the results obtained with the Genetic Algorithm show that a good feasible solution is possible to achieve with the new automatic loop-shaping strategy. At the same time it is important to accentuate that the strategy is sensitive to the fitness function coefficients. Depending on their adjustment, a feasible or an unfeasible solution is achieved. In that point, it is possible to conclude that intuitive considerations in the search may be very useful, and the adjustment of the fitness evaluation function is of great importance for the success. Defining functions less sensitive to coefficient value should be an interesting future work because this is, in fact, one of the main limitations of the Evolutionary and Genetic Algorithms.

Although not always the designer can achieve a feasible solution to the problem, it can represent a first approach to the final solution, in the sense that the obtained controller may be modified until getting a feasible one.

To sum up, this methodology based on Evolutionary Algorithms and Genetic Algorithms can contribute to improve QFT controller design, alleviating much of the manual work required at present.

6. REFERENCES

[1] Ballance, D.J. and P.H. Gawthrop, Control systems design via a Quantitative Feedback Theory approach, *Proc. of the IEE Conference Control´91*, Edinburgh, UK, pp. 476-480, 1991.

[2] Blakelock, J., H., *Automatic Control of Aircraft and Missiles* (Wiley, New York 1965).

[3] Borghesani, C., Y. Chait, O. Yaniv, *Quantitative Feedback Theory Toolbox v2.0 - For use with MATLAB* (Terasoft, 2003).

[4] Brindle, A., *Genetic Algorithms for Function Optimization*, Doctoral Dissertation, University of Alberta, Edmonton, 1981.

[5] Bryant, G.F. and G.D. Halikias, Optimal loop-shaping for systems with large parameter uncertainty via linear programming. *International Journal of Control*, 62, pp.557-568, 1995.

[6] Chait, Y., Q. Chen, C. V. Hollot, Automatic Loop-Shaping of QFT Controllers via Linear Programming, *ASME Journal of Dynamic Systems, Measurement and Control*, 121, pp. 351-357, 1999.

[7] Chait, Y. and O. Yaniv, Multi-input/Single-output computer-aided control design using the Quantitative Feedback Theory, *Int. Journal of Robust and Nonlinear Control*, 3, pp. 47-54, 1993.

[8] Chen, W., D. J. Ballance, W. Feng, Y. Li, Genetic Algorithm Enabled Computer-Automated Design, *Procedings of the IEEE International Symposium on Computer Aided Control System Design*, Kohala Coast-Island of Hawaii, Hawaii, USA, 1999, pp. 492-497.

[9] Chen, W., D.J. Balance and Y. Li, Automatic Loop-shaping in QFT using Genetic Algorithms, *Proceedings of the 3rd Asia-Pacific Conference on Control and Measurement*, pp. 63-67, 1998.

[10] Garcia-Sanz, M. and J.A. Oses, Evolutionary Algorithms for Automatic tuning of QFT Controllers, *Proceedings of the 23rd IASTED International Conference Modelling, Identification and Control*, Grindelwald, Switzerland, 2004.

[11] Garcia-Sanz, M. and J.C. Guillen, Automatic Loop-shaping of QFT Robust Controllers via Genetic Algorithms, *3rd IFAC Symposium on Robust Control Design*, Prague, Czech Republic, 2000.

[12] Garcia-Sanz, M. and M.Barreras, Non-diagonal QFT controller design for a 3-input 3-output industrial Furnace, *International Journal of Dynamic Systems, Measurement and Control, ASME*, USA, June 2006, Vol. 128, pp. 319-329.

[13] Garcia-Sanz, M., I. Eguinoa, M. Barreras and S. Bennani, Non-diagonal MIMO QFT Controller Design for Darwin-type Spacecraft with large flimsy appendages, *Journal of Dynamic Systems, Measurement and Control, ASME*, Vol. 130, pp. 011006-1:011006-15, January 2008, USA.

[14] Garcia-Sanz, M. and J. Elso, Beyond the linear limitations by combining Switching & QFT. Application to Wind Turbines Pitch Control Systems, Special Issue: Wind Turbines: New Challenges and Advanced Control Solutions, *International Journal of Robust and Non-Linear Control*, Wiley, Vol. 18, No. 12, 2008.

[15] Garcia-Sanz, M., I. Eguinoa and S. Bennani, Non-diagonal MIMO QFT controller design reformulation, *International Journal of Robust and Non-Linear Control*, published online in Wiley Interscience, DOI: 10.1002/rnc.1362, July 2008.

[16] Garcia-Sanz, M., I. Egaña and J. Villanueva, Quantitative Multivariable Feedback Design for a SCARA Robot Arm, *5th International Symposium on Quantitative Feedback Theory QFT and Robust Frequency Domain Methods*, pp. 67-72, August 2001, Public University of Navarre, Pamplona, Spain, ISBN: 84-95075-56-3.

[17] Garcia-Sanz, M., M. Barreras, I. Egaña and C.H. Houpis, External Disturbance Rejection in Uncertain MIMO systems with QFT non-diagonal controllers, *6th International Symposium on Quantitative Feedback Theory QFT and Robust Frequency Domain Methods*, December 2003, Cape Town, South Africa.

[18] Garcia-Sanz, M. and M. Barreras, Multivariable QFT controller design for heat exchangers of power plants, *2nd International Conference on Renewable Energy and Power Quality ICREPQ'04*, April 2004, Barcelona, Spain.

[19] Garcia-Sanz, M., I. Eguinoa, E. Ayesa, and C. Martin, Non-diagonal multivariable robust QFT control of a wastewater treatment plant for simultaneous nitrogen and phosphorus removal, *Robust Control Design Conference, ROCOND'06, IFAC*, 2006, Toulouse, France.

[20] Garcia-Sanz, M., I. Eguinoa and S. Bennani, Full-Matrix Inverse-Based MIMO QFT Control Design For Spacecraft Flying in Formation, *17th IFAC Conference on Aerospace*, Toulouse, France, 2007.

[21] Garcia-Sanz, M. and C. Molins, QFT Robust Control of a Vega-type Space Launcher, *16th Mediterranean Conference on Control and Automation, MED'08*, Corcega, France, June 2008.

[22] Gera, A. and I.M. Horowitz, Optimization of the loop transfer function, *International Journal of Control*, 31, pp. 389-398, 1992.

[23] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison-Wesley, 1989).

[24] Gutman, P.O., *Qsyn – The Toolbox for Robust Control Systems Design for use with Matlab, User's Guide*, (El-Op Electro-Optics Industries Ltd, Rehovot, 1996).

[25] Holland, J.H., *Adaptation in Natural and Artificial Systems*, (Ann Arbor: University of Michigan Press, 1975).

[26] Horowitz, I. M., Optimum loop transfer function in single-loop minimum phase feedback systems, *International Journal of Control*, 22, 1973, pp. 97-113.

[27] Horowitz, I. M., Survey of quantitative feedback theory (QFT), *International Journal of Control*, 53 (2), 1991, pp. 255-291.

[28] Horowitz, I. M., Quantitative Feedback Design Theory-QFT (Vol.1), QFT Press, Boulder, Colorado, USA, 1993.

[29] Horowitz, I. M. and M. Sidi, Synthesis of Feedback Systems with Large Plant Ignorance for Prescribed Time-Domain Tolerances, *International Journal of Control*, vol. 16, n. 2, 1972, pp. 287-309.

[30] Houpis, C.H., S. J. Rasmussen and M. Garcia-Sanz *Quantitative Feedback Theory. Fundamentals and Applications*. 2nd Edition (Taylor & Francis, 2006).

[31] Houpis, C.H. and R.R. Sating (1997), *MIMO QFT CAD Package (Ver.3)*, Int. J. Control, Vol. 7, No. 6, pp. 533-549.

[32] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs* (3rd Revised and Extended Edition. Heidelberg: Springer-Verlang Berlin, 1996).

[33] Nataraj, P.S.V. and S. Tharewal, An Interval Analysis Algorithm for Automated Controller Synthesis in QFT Designs, *ASME J. of Dynamic Systems, Measurement and Control*, 129, pp. 311-321, 2007.

[34] Sarker, R., M. Mohammadian, X. Yao, *Evolutionary Optimization*, (Kluwer Academic Publishers, 2002).

[35] Sating, R.R., (1992). *Development of an Analog MIMO Quantitative Feedback Theory (QFT) CAD Package*, MS Thesis, AFIT/GE/ENG/92J-04, Air Force Institute of Technology, Wright Patterson AFB, OH.

[36] Sating, R.R., I.M. Horowitz and C.H. Houpis (1993), *Development of a MIMO QFT CAD Package (Ver.2)*, Air Force Institute of Technology, Wright-Patterson AFB, OH 45433, USA, American Control Conference.

[37] Skogestad, S., I. Postlethwaite, *Multivariable Feedback Control. Analysis and Design*. 2nd Edition.(Wiley & Sons, UK , 2005)

[38] Thompson, D.F. and O.D.I. Nwokah, Analytic loop shaping methods in Quantitative Feedback Theory, *ASME J. of Dynamic Systems, Measurement and Control*, 116, pp. 169-177, 1994.

[39] Thompson, D.F., Gain-bandwidth optimal design for the New Formulation Quantitative Feedback Theory, *ASME J. of Dynamic Systems, Measurement and Control*, 120, pp. 401-404, 1998.

[40] Yaniv, O., *Quantitative Feedback Design of Linear and Non-linear Control Systems*. (Massachusetts, USA: Kluver Academic Publishers).

7. APPENDIX A: MATLAB FILES FOR THE STRATEGY 1 (EVOLUTIONARY ALGORITHMS)

```
% AUTOLOOPSHAPING_ES Program to synthesize optimal controllers with an
%                    initial structure which can be modified by means of
%                    optimal Hankel norm approximation If any modification
%                    occurs the program synthesizes the optimal controllerfor
%                    the new structure.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

%----------------------Plant templates and bounds-------------------------

% Before executing this program the user has to define the plant and the
% specifications at Tmpl_Bnds.m, as well as all the parameters needed for
% the search strategy at configuration.m

%--------------------Templates and bounds creation-------------------------

[bdb, np, dp] = Tmpl_Bnds();

%----------------------Parameters configuration---------------------------

% Configuration of all the parameters needed for the search strategy.

configuration;

%----------------------Automatic loop-shaping-----------------------------

z=1;
nrz0=nrz;
ncz0=ncz;
npr0=npr;
npc0=npc;
k=npr+2*npc-1;

while k>=1 % Loop to calculate the controllers while the structure varies.

    % ------------------------Evolutive Algorithm---------------------

    EvAl;

    % The results obtained in each iteration are stored in vectors
    % (gain,realzeros,complexzeros...)

    gain(z,:)=kg;

    if nrz==0
        realzeros(z,:)=0;
    else
        realzeros(z,1:nrz)=rz;
    end
```

```
if ncz==0
    complexzeros(z,:)=0;
else
    complexzeros(z,1:2*ncz)=cz;
end
if npr==0
    realpoles(z,:)=0;
else
    realpoles(z,1:npr)=pr;
end
if npc==0
    complexpoles(z,:)=0;
else
    complexpoles(z,1:2*npc)=pc;
end

fcost(z,:)=y;
z=z+1;


% --------------------Model reduction---------------------
%
% Now the obtained controller is reduced to a k-th order model by means
% of the optimal Hankel norm approximation.

[numC, denC] = TG(kg,cz,rz,pc,pr);
sys=tf(numC,denC);
sys=ss(sys);
n=size(sys.A,1);
nu=size(sys,2);
[sysc,cinfo]=ncfmr(sys,n);   % or obtain coprime factors of system
sysch=hankelmr(cinfo.GL,k);  % or optimal Hankel norm approximation
syschm=minreal(inv(sysch(:,nu+1 ...%and obtain k-th order model
    :end))*sysch(:,1:nu));

[Z,P,K]=ZPKDATA(syschm,'v'); % Gain, poles and zeros of the reduced
                             % model.

% Storage of the gain, poles and zeros of the reduced model in the
% correspondent vectors (gain, complexzeros, realzeros,...).

nrz=0;
ncz=0;
npr=0;
npc=0;
realzeros(z,1:nrz)=0;
complexzeros(z,1:2*ncz)=0;
realpoles(z,1:npr)=0;
complexpoles(z,1:2*npc)=0;

if ~isempty(Z)
    ii=1;
    ri=1;
    ci=1;
    while ii<=length(Z)
        if imag(Z(ii))~=0
            complexzeros(z,ci)=abs(real(Z(ii)));
            complexzeros(z,ci+1)=abs(imag(Z(ii)));
```

```
            ii=ii+2;
            ci=ci+2;
            ncz=ncz+1;
        else
            realzeros(z,ri)=abs(Z(ii));
            ii=ii+1;
            ri=ri+1;
            nrz=nrz+1;
        end
    end
end


if ~isempty(P)
    jj=1;
    rj=1;
    cj=1;
    while jj<=length(P)
        if imag(P(jj))~=0
            complexpoles(z,cj)=abs(real(P(jj)));
            complexpoles(z,cj+1)=abs(imag(P(jj)));
            jj=jj+2;
            cj=cj+2;
            npc=npc+1;
        else
            realpoles(z,rj)=abs(P(jj));
            jj=jj+1;
            rj=rj+1;
            npr=npr+1;
        end
    end
end

kg=K;
rz=realzeros(z,1:nrz);
cz=complexzeros(z,1:2*ncz);
pr=realpoles(z,1:npr);
pc=complexpoles(z,1:2*npc);
Kg = K_cor2(kg,cz,rz,pc,pr);
gain(z,:)=Kg;

z=z+1;
k=npr+2*npc-1;
Range_z(:,(nrz+2*ncz+1):end)=[];
Range_p(:,(npr+2*npc+1):end)=[];

if k==0 % This loop is to calculate the 1st order controller with the
        % search strategy.
    EvAl;
    gain(z,:)=kg;

    if nrz==0
        realzeros(z,:)=0;
    else
        realzeros(z,1:nrz)=rz;
    end
    if ncz==0
        complexzeros(z,:)=0;
    else
```

```
            complexzeros(z,1:2*ncz)=cz;
        end
        if npr==0
            realpoles(z,:)=0;
        else
            realpoles(z,1:npr)=pr;
        end
        if npc==0
            complexpoles(z,:)=0;
        else
            complexpoles(z,1:2*npc)=pc;
        end

        fcost(z,:)=y;
        z=z+1;
    end
end


% Once the controllers have been obtained some optional tasks can be
% performed

clear cp;
[r,t]=size(np);
for i=1:1:r
    cp(1,1,i)=tf(np(i,:),dp(i,:));
end


% Storage of all the obtained controllers

controllers;

% Menu to perform several tasks with the obtained controllers.

wcheck=logspace(-4,4,100);save wcheck; % frequencies for the chksiso in
menuplot
wloop=logspace(-2,4,200); save wloop;% frequencies for the lpshape in menuplot
menuplot;


% BVALID Function to check the bounds fulfilment at the frequencies of
%        operation.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function [v, vvar] = BValid(kg,cz,rz,pc,pr,bdb_a,bdb_b,w,nP0,dP0)

% ------------- Bounds restrictions -----------------
Vw = [];
for k = 1:length(w)
  [mod, ph] = TLo(kg,cz,rz,pc,pr,nP0,dP0,w(k));
  mod = log10(mod);
  qu = ilingrad(log10(bdb_a(:,k)), ph);
  ql = ilingrad(log10(bdb_b(:,k)), ph);
```

```
  Vu = max(qu - mod, 0);
  Vl = max(mod - ql, 0);
  if qu >= ql
    Vw = [Vw, min(Vu, Vl)];
  else
    Vw = [Vw, max(Vu, Vl)];
  end;
end;

% ---------------- Sum and variance calculation -----------------
v = 0;           % Bounds violation
vvar = 0;        % Violation variance
v = sum(Vw.^2);
if length(Vw) > 2
    vvar = var(Vw);
end;


% CONFIGURATION Configuration of all the parameters needed for the
%               automatic loop-shaping.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

% -------------------- Starting parameters for G(s) ---------------

nrz = 2; % Number of real zeros.
ncz = 0; % Number of complex zeros.
npr = 1; % Number of real poles.
npc = 1; % Number of complex poles.

% ------------------ Multi-objective function coefficients -----------

a1 = 0.001;    % Gain objective.
a2 = 0.001;    % Closeness to UHFB objective.
a3 = 2;        % Decreasing phase objective.

r1 = [5, 18];  % Bounds non-fulfillment penalty.
r2 = [4, 16];  % Instability penalty.
r3 = [4, 16];  % Non-decreasing |Lo(s)| penalty.

% ------------ Evolutive algorithm parameters -----------------

lambda = 0.2;  % Mutation amplitude.

prob_mut = 0.08;    % Mutation probability.
prob_cros = 0.25;   % Cross probability.

% ---------------- Number of individuals in the population -----------

N_vectors = 80;       % Number of individuals in each generation.
N_generations = 300;  % Number of generations.

% -------------------- Poles and zeros ranges --------------------

% Zeros range.
```

```
Range_z = [0.01 0.01 ; ...
           400 400 ];

% Poles range.
Range_p = [0.01 0.01 0.01 ; ...
           1500 1500 1500 ];

% Gain range.
Range_k = [0.01 1500];

% ----------------- Range union -------------------

Range = [Range_k', Range_z, Range_p];


% CONTROLLERS Program to store all the controllers that have been obtained.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

for h=1:1:z-1   % for the z-1 controllers that have been obtained.
    mrealpoles=1;  % Initialize parameters
    mrealzeros=1;
    mcomplexpoles=1;
    mcomplexzeros=1;
    for hz=1:1:numel(find(realzeros(h,:)>0))  % for the nzr real zeros
        if realzeros(h,hz)~=0
            mrealzeros=conv(mrealzeros,[1/realzeros(h,hz) 1]);
        end
    end
    for hp=1:1:numel(find(realpoles(h,:)>0))  % for the npr real poles
        if realpoles(h,hp)~=0
            mrealpoles=conv(mrealpoles,[1/realpoles(h,hp) 1]);
        end
    end
    for hcz=1:2:numel(find(complexzeros(h,:)>0))-1 %for the complex zeros
        if complexzeros(h,hcz+1)~=0
        omegan1=sqrt((complexzeros(h,hcz)^2)+...
            (complexzeros(h,hcz+1)^2));
        mcomplexzeros=conv(mcomplexzeros,[1/(omegan1^2) ...
            2*complexzeros(h,hcz)/(omegan1^2) 1]);
        end
    end
    for hpc=1:2:numel(find(complexpoles(h,:)>0))-1 %for the complex poles
        if complexpoles(h,hpc+1)~=0
        omegan2=sqrt((complexpoles(h,hpc)^2)+...
            (complexpoles(h,hpc+1)^2));
        mcomplexpoles=conv(mcomplexpoles,[1/(omegan2^2) ...
            2*complexpoles(h,hpc)/(omegan2^2) 1]);
        end
    end
    numG=conv(gain(h),conv(mrealzeros,mcomplexzeros));
    denG=conv(mrealpoles,mcomplexpoles);
    G(h)=tf(numG,denG);  % To store the z-1 controllers

end
```

```
% COSTS Cost function for the G(s) controller, considering complex poles
%       and zeros. Function to evaluate the cost function.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function [y, v] = costS(X,nrz,ncz,npr,npc,bdb_a,bdb_b,w,nP0,dP0)

% ------------- Poles and zeros extraction from X vector ------------
% Structure:
%        [gain,   complex zeros,   real zeros,
%               complex poles,   real poles]
%

kg = X(1);
cz = X(2:1+2*ncz);
rz = X(2+2*ncz:1+2*ncz+nrz);
pc = X(nrz+2*ncz+2:1+nrz+2*ncz+2*npc);
pr = X(2+nrz+2*ncz+2*npc:1+nrz+2*ncz+2*npc+npr);

% Bounds fulfilment
v = BValid(kg,cz,rz,pc,pr,bdb_a,bdb_b,w,nP0,dP0);

% System stability
st = stable(kg,cz,rz,pc,pr,nP0,dP0);

% Controller with decreasing |Lo(s)|
ds = chkDescen(kg, cz,rz,pc,pr, nP0, dP0, [0.1 1000]);

% Controller tangent to UHFB
mf = LUHFB(kg,cz,rz,pc,pr,bdb_a,bdb_b,w,nP0,dP0,100);

% Controller with decreasing phase
dsph = chkPhDes(kg, cz,rz,pc,pr, nP0, dP0, w);

% Cost vector
y = [kg^2, mf, dsph, v, st, ds];


% CHKDESCEN To check if a controller produces a decreasing |Lo(s)|.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function y = chkDescen(kg, cz, rz, pc, pr, nP0, dP0, w)

% Open loop function Lo(s)
 [numG, denG] = TG(kg,cz,rz,pc,pr);
T2 = conv(numG, nP0);
T1 = conv(denG, dP0);

P = tf(T2,T1);
Mod = 20*log10(abs(P));
```

```
wdB = log10(freq);
for i=2:length(wdB)
    k(i-1) = (Mod(i) - Mod(i-1))/(wdB(i) - wdB(i-1));
end;

y = 0;
if max(k) > 0
    y = 1;
end;


% CHKPHDES Function to check if the phase is always decreasing.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------

function y = chkPhDes(kg, cz, rz, pc, pr, nP0, dP0, w)

% Lo(s) transfer function

[numG, denG] = TG(kg,cz,rz,pc,pr);
T2 = conv(numG, nP0);
T1 = conv(denG, dP0);

% Check if the phase increases in any point.

P = tf(T2,T1);
Ph = angle(P);
for i=2:length(w)
    k(i-1) = Ph(i) - Ph(i - 1);
    if k(i - 1) < 0
        k(i - 1) = 0;
    end;
end;

y = sum(k);


% DISORDER Function to disorder the rows of a matrix.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-----------------------------------------------------------------

function Y = disorder(X)

Y = [];
[r,c] = size(X);
if r > 0
    for i=1:r
        [rr, cc] = size(Y);
        if rr > 0
            pos = round((rr-1)*rand(1)+1);
        else
            pos = 1;
        end;
```

```
        Y = [Y(1:pos-1,:); X(i,:); Y(pos:rr,:)];
    end;
end;


% DOMINANCE Function to know the dominance of the objectives or
%           constraints in the fitness function.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function [dt1, dt2] = dominance(XCost,a1,a2,a3,r1,r2,r3)

% Cost
% [kg, Phase_dif, Des_phase, Bounds, Stability, Des_mod]

[r,c] = size(XCost);
parameters = [a1, a2, a3, r1, r2, r3];
for i=1:r
    dr1 = 0;
    dr2 = 0;
    dr3 = 0;
    if XCost(i,4) > 0
        dr1 = 1;
    end;
    if XCost(i,5) > 0
        dr2 = 1;
    end;
    if XCost(i,6) > 0
        dr3 = 1;
    end;

    XCost_t(i,:) = XCost(i,:).*parameters;
    XCost_t(i,:) = XCost_t(i,:) + [0, 0, 0, 1, 1, 0];
    XCost_t(i,:) = [XCost_t(i,1:3), [dr1, dr2, dr3].*XCost_t(i,4:6)];
end;

F = sum(XCost_t(:,4:6),2);
Q = sum(XCost_t(:,1:3),2);

mu = F./Q;

nf = 0;
nq = 0;
for i=1:length(mu)
    if mu(i) > 1
        nf = nf + 1;
    else
        nq = nq + 1;
    end;
end;

dt2 = mu(1);
dt1 = [nq, nf];
```

```
% EvAl  Matlab script to study the evolutive strategies.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

% ----------------- Initial population --------------------

X = [];
for i=1:N_vectors
  X_r = [];
  for j=1:length(Range)
      X_r = [X_r, Range(1,j) + rand(1)*(Range(2,j) - Range(1,j))];
  end;
  X = [X; X_r]; % matrix that stores in each row an individual of the
                % initial population.
end;

% -------- Bounds into non-logarithm scale ---------

[r, c] = size(bdb(1:length(bdb)-2,:));
bdb_a = 10.^(bdb(1:r/2,:)/20);
bdb_b = 10.^(bdb(r/2+1:r,:)/20);

w = bdb(length(bdb)-1,:); % bounds frequency vector

% ------------------- Nominal Plant ---------------------------

nompt = 1;          % The nominal plant is always in the first row.
nP0 = np(nompt,:);
dP0 = dp(nompt,:);

% ------------------ Evolutive algorithm ------------

[best_vector, best_ac, dist_ac, X, dominant] = ...
    evolStr(X,N_generations,prob_mut, prob_cros,Range,nrz,ncz,npr,npc,...
    a1,a2,a3,r1,r2,r3,lambda,bdb_a,bdb_b,w,nP0,dP0);

% --------------- Comparisons ----------------

y = costS(best_vector,nrz,ncz,npr,npc,bdb_a,bdb_b,w,nP0,dP0);
message('Best controller found (X)','Cost:',best_vector,nrz,ncz,npr,npc,y);

% ----------------- Controller validity ----------------------

kg = best_vector(1);
cz = best_vector(2:1+2*ncz);
rz = best_vector(2+2*ncz:1+2*ncz+nrz);
pc = best_vector(nrz+2*ncz+2:1+nrz+2*ncz+2*npc);
pr = best_vector(2+nrz+2*ncz+2*npc:1+nrz+2*ncz+2*npc+npr);
st = stable(kg,cz,rz,pc,pr,nP0,dP0);
if st > 0
    disp('unstable controller');
else
    disp('stable controller');
end
```

```
% EVOLSTR Function to implement a simple evolutive strategy.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-----------------------------------------------------------------

function [best_vector, best_ac, dist_ac, X, dominant] = ...
    evolStr(X,M_count,prob_mut, prob_cros, Range,nrz,ncz,npr,npc,...
    a1,a2,a3,ar1,ar2,ar3,lambda,bdb_a,bdb_b,w,nP0,dP0)

r1 = ar1(1);
r2 = ar2(1);
r3 = ar3(1);

rand('state',0);
N_vectors = size(X(:,1));

% ---------- Evolutionary Estrategy (ES) --------------

disp('Start of the evolutive method (ES) ... ');
disp(' ');

% -------- Acumulators setup --------------

dist_ac = [];
best_vector = [];
best_ac = [];
mut_number = 0;
mut_success = 0;
dominant = [];

% ------------- Best vector of the initial population ---------

F = [];
D = [];
for i=1:N_vectors
  [F(i,:), D(i)] = costS(X(i,:),nrz,ncz,npr,npc,bdb_a,bdb_b,w,nP0,dP0);
end;
[X, indexF, X_cost2]=selection(X,F,a1,a2,a3,r1,r2,r3,r1,r2,r3,0,N_vectors);

best_vector = X(1,:);
best_ac = [best_ac; best_vector];
dist_ac = X_cost2(1,4);
score = indexF(1,:);
[dt1, dt2] = dominance(X_cost2,a1,a2,a3,r1,r2,r3);
dominant = [dominant; [dt1, dt2]];

% ------------- Operation cycle --------------------------

rp = [ar1; ar2; ar3];
count = 0;           % Iterations accountant
while (count < M_count)

% ----------------- Penalties establishment ------------
```

```
  r_new = rp(:,1) + (rp(:,2) - rp(:,1))*count/M_count;
  nr1 = r_new(1);
  nr2 = r_new(2);
  nr3 = r_new(3);

% ------------------- Crossover ------------------------
  X_cost = [];
  X_cross = [];      % Population for the crossover.
  X_no_cross = [];
  for i=1:N_vectors
    if rand(1) < prob_cros
      X_cross = [X_cross; X(i,:)];
    else
      X_no_cross = [X_no_cross; X(i,:)];
    end;
  end;

  mark = 0;   % If mark == 1, we add one element to have an even number.
  [r,c] = size(X_cross);
  if r > 0
    if rem(length(X_cross(:,1)), 2) > 0
      X_cross = [X_cross; X_cross(length(X_cross(:,1)),:)];
      mark = 1;
    end;
    X_cross=disorder(X_cross);% Disorder the rows of the matrix.
    X_cross_new = [];
    for i=1:2:length(X_cross(:,1))-1
      cross_element_1 = [];  % Child 1
      cross_element_2 = [];  % Child 2
      for k=1:length(X(1,:))
         if rand(1) < 0.3
          a = rand(1);
        else
          a = round(rand(1));
        end
        cross_element_1 = [cross_element_1, a*X_cross(i,k) + ...
            (1-a)*X_cross(i+1,k)];
        cross_element_2 = [cross_element_2, (1-a)*X_cross(i,k) + ...
            a*X_cross(i+1,k)];
      end;
      % Join the parents with their children and select the two bests.
      X_sub_cross = [X_cross(i,:); X_cross(i+1,:); cross_element_1; ...
          cross_element_2];

      F = [];
      D = [];
      for i=1:4
          [F(i,:) D(i)] = costS(X_sub_cross(i,:),nrz,ncz,npr,npc,bdb_a,...
              bdb_b,w,nP0,dP0);
      end;
      [X_cross2, I_cross, X_cost2] = selection(X_sub_cross, F, a1,a2,...
          a3,r1,r2,r3,nr1,nr2,nr3,2,2);
      X_cross_new = [X_cross_new; X_cross2];
      X_cost = [X_cost; X_cost2];
    end;
  end;

  % ------------- Mutation --------------------
```

```
X_mut = [];
X_no_mut = [];
[r, c] = size(X);
for i = 1:r
  if rand(1) < c*prob_mut
    if rand(1) > 0.3
        % Only one parameter mutates.
        pos = round((c-1)*rand(1)+1);  % Mutation position.
        new_value = randomr(X(i,pos), Range(1,pos),Range(2,pos), lambda);
        Mut_vector = X(i,:);
        Mut_vector(pos) = new_value;
    else
        % All the parameters mutate.
        for k = 1:c
            Mut_vector(k) = randomr(X(i,k),Range(1,k),Range(2,k),lambda);
        end;
    end;

    F = [];
    D = [];
    [F(1,:),D(1)]=costS(Mut_vector,nrz,ncz,npr,npc,bdb_a,bdb_b,w,nP0,dP0);
    [F(2,:),D(2)]=costS(X(i,:),nrz,ncz,npr,npc,bdb_a,bdb_b,w,nP0,dP0);
    [X_mut2,I_mut,X_cost2]=selection([Mut_vector;X(i,:)],F,a1,a2,a3,...
        r1,r2,r3,nr1,nr2,nr3,1,1);
    X_mut = [X_mut; X_mut2];
    X_cost = [X_cost; X_cost2];

    if X_mut2 == Mut_vector
        mut_success = mut_success + 1;
    end;
  else
    X_no_mut = [X_no_mut; X(i,:)];
  end;
end;

[r, c] = size(X_mut);
mut_number = mut_number + r;

% ---------- Evolution of mutation range -------

if rem(count, 5) == 0
  disp(sprintf('Step number %d. Mutation success %2.3f',count,...
      mut_success/mut_number));
  vari_s = [''];
  param_s = [''];
  score_s = [''];
  dominant_s = [''];
  for i=1:length(best_vector)
      param_s = sprintf('%s %4.4f', param_s, best_vector(i));
      vari_s = sprintf('%s %2.4f', vari_s, sqrt(var(X(:,i))));
      [sr, sc] = size(score);
  end;
  for k=1:sc
      score_s = sprintf('%s %4.3f ', score_s, score(sr,k));
  end;
  [dr,dc] = size(dominant);
  for k=1:dc
      dominant_s = sprintf('nq = %d, nf = %d, %3.4f',dominant(dr,1),...
          dominant(dr,2), dominant(dr,3));
```

```
    end;
    disp(sprintf('Vector:     %s', param_s));
    disp(sprintf('Stand.Dev.: %s', vari_s));
    disp(sprintf('Classification: [%s]',score_s));
    disp(sprintf('Population balance.: %s', dominant_s));
    disp('  ');
    mut_success = 0;
    mut_number = 0;
  end;

  % ------------------ Selection ----------------------
  % X = [X_no_mut; X_cross_new; X_mut].

  X = [X_cross_new; X_mut];
  nn = size(X(:,1));

  X = [X; X_no_mut];

  [mr, mc] = size(X_no_mut);
  F = [];
  for i=1:mr
    F(i,:) = costS(X_no_mut(i,:),nrz,ncz,npr,npc,bdb_a,bdb_b,w,nP0,dP0);
  end;
  % F = [F; X_cost].
  F = [X_cost; F];

  [X_new, I_new, Cost_new] = selection(X,F,a1,a2,a3,r1,r2,r3,nr1,nr2,...
      nr3,nn,N_vectors);

  % Population update.
  X = X_new;

  % Acumulators update.
  best_vector = X(1,:);
  dist_ac = [dist_ac, Cost_new(1,4)];
  score = [score; I_new(1,:)];
  best_ac = [best_ac; best_vector];

  [dt1, dt2] = dominance(Cost_new, a1,a2,a3,nr1,nr2,nr3);
  dominant = [dominant; [dt1, dt2]];

  % Penalties update.
  r1 = nr1;
  r2 = nr2;
  r3 = nr3;

  % Accountant increment.
  count = count + 1;

end;

% ILINGRAD Function for the linear interpolation of a function (Fin) with
%          repsect to the phase from 0 to -360 degrees, in a x point.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-----------------------------------------------------------------------
```

```
function result = ilingrad(Fin, x)

% Number of points of Fin
N = length(Fin);

% Function vector invertion
for i=1:N
  F(i) = Fin(N+1-i);
end;

% Phase division in degrees.
delta = -360/(N-1);
na = N - floor(x/delta);
nb = N - ceil(x/delta);

% Linear interpolation
if na == nb
  result = F(nb);
else
  result = F(nb) + (N - x/delta - nb)*(F(na) - F(nb));
end;

% Maximum and minimum value limitation.
if result > 0
  if result < 0.00001
    result = 0.00001;
  end;
else
  if result > -0.00001
    result = -0.00001;
  end;
end;


% ILINMOD Function for the linear interpolation of a function (Fin) with
%         respect to the module.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function result = ilinmod(Fin, x)

% Number of points of Fin
N = length(Fin(1,:));
Fmod = Fin(1,:);
FPh = - Fin(2,:);

if (x < min(Fmod)) | (x > max(Fmod))
    result = -1;
else
    i = 1;
    while x >= Fmod(i)
        i = i + 1;
    end;
    m1 = Fmod(i - 1);
    ph1 = FPh(i - 1);
```

```
    i = N;
    while x <= Fmod(i)
        i = i - 1;
    end;
    m2 = Fmod(i + 1);
    ph2 = FPh(i + 1);

    if m1 < m2
        result = ph1 + (x - m1)*(ph2 - ph1)/(m2 -m1);
    else
        result = ph1;
    end;
end;



% K_COR Function to correct the value of Kg due to the way the poles and
%       zeros are defined.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------


function Kg = K_cor(kg,cz,rz,pc,pr)

Kg = kg;

numG = kg;
denG = 1.0;

% Complex zeros

for j=1:2:numel(find(cz(:)>0))
    Kg = Kg / (cz(j)^2 + cz(j+1)^2);
end


% Real zeros
for j=1:numel(find(rz(:)>0))
    Kg = Kg / rz(j);
end;

% Complex poles
for j=1:2:numel(find(pc(:)>0))
    Kg = Kg * (pc(j)^2 + pc(j+1)^2);
end;

% Real poles
for j=1:numel(find(pr(:)>0))
    Kg = Kg * pr(j);
end;
```

```
% K_COR2 Function to correct the value of Kg due to the way the poles and
%        zeros are defined.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------


function Kg = K_cor2(kg,cz,rz,pc,pr)

Kg = kg;

numG = kg;
denG = 1.0;

% Complex zeros

for j=1:2:numel(find(cz(:)>0))
    Kg = Kg * (cz(j)^2 + cz(j+1)^2);
end


% Real zeros

for j=1:numel(find(rz(:)>0))
    Kg = Kg * rz(j);
end;

% Complex poles

for j=1:2:numel(find(pc(:)>0))
    Kg = Kg / (pc(j)^2 + pc(j+1)^2);
end;


% Real poles

for j=1:numel(find(pr(:)>0))
    Kg = Kg / pr(j);
end;

% LUHFB  Function to evaluate how tangent is Lo to the UHFB.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function ph_ac = LUHFB(kg,cz,rz,pc,pr,bdb_a,bdb_b,w,nP0,dP0,w_ref)

% ------------- Bounds restrictions -----------------
ph_ac = 0;
for k = 1:length(w)
  [mod, ph] = TLo(kg,cz,rz,pc,pr,nP0,dP0,w(k));
  mark = min(20*log10(bdb_b(:,k)));
  mod = 20*log10(mod);
```

```
  if (mark < 320) & (w(k) >= w_ref)
      y = uph(20*log10(bdb_a), 20*log10(bdb_b));
      if mod > min(y(1,:))
          ph_b = ilinmod(y, mod);
          if ph_b > 0
              diff = abs(ph_b - abs(ph));
              if diff > 5
                  ph_ac = ph_ac + diff;
              end;
          end;
      end;
  end;
end;


% MENUPLOT Menu to choose the task to do with the results obtained from the
%          automatic loop-shaping.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

k=menu('Choose an option','step','bode','nichols','nyquist','chksiso');

switch k
    case 1
        for h=1:1:z-1
            figure(2)
            step(cp(1,1,1)*G(h));
            grid on;
            set(gca, 'FontSize',[5])
            axis([0 70 0 200])
            hold all
        end
    case 2
        for h=1:1:z-1
            figure(2)
            bode(cp(1,1,1)*G(h),w);
            grid on;
            set(gca, 'FontSize',[5])
            axis([0.001 10000 -180 -45])
            hold all
        end
    case 3
        load wloop;
        for h=1:1:z-1
            lpshape(wloop, bdb, cp(1,1,1), G(h), []);
        end
    case 4
        for h=1:1:z-1
            figure(2)
            nyquist(cp(1,1,1)*G(h));
            grid on;
            set(gca,'FontSize',[5])
            axis([-6 6 -3 3])
            hold all
        end
    case 5
        load wcheck;
```

```
        load W1
        load W7
        for h=1:1:z-1
            if ~isempty(W1)
            chksiso(1,wcheck,W1,cp,0,G(h),1,1);
            title(['bound type 1 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W2)
            chksiso(2,wcheck,W2,cp,0,G(h),1,1);
            title(['bound type 2 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W3)
            chksiso(3,wcheck,W3,cp,0,G(h),1,1);
            title(['bound type 3 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W4)
            chksiso(4,wcheck,W4,cp,0,G(h),1,1);
            title(['bound type 4 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W5)
            chksiso(5,wcheck,W5,cp,0,G(h),1,1);
            title(['bound type 5 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W6)
            chksiso(6,wcheck,W6,cp,0,G(h),1,1);
            title(['bound type 6 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W7)
            chksiso(7,wcheck,W7,cp,0,G(h),1,1);
            title(['bound type 7 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W8)
            chksiso(8,wcheck,W8,cp,0,G(h),1,1);
            title(['bound type 8 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W9)
            chksiso(9,wcheck,W9,cp,0,G(h),1,1);
            title(['bound type 9 with the ',num2str(h),'th controller']);
            grid on;
            end
        end
end


% MESSAGE Function to show the gain, the zeros, the poles and the cost.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-----------------------------------------------------------------------

function result = message(m1,m2,x,nrz,ncz,npr,npc,cost)
```

```
disp(m1);

kg = x(1);
cz = x(2:1+2*ncz);
rz = x(2+2*ncz:1+2*ncz+nrz);
pc = x(nrz+2*ncz+2:1+nrz+2*ncz+2*npc);
pr = x(2+nrz+2*ncz+2*npc:1+nrz+2*ncz+2*npc+npr);

khf = K_cor(kg, cz, rz, pc, pr);

S = sprintf('  kg = %4.2f <HFG = %4.2f>\n  Zeros: ', kg, khf);
for i=1:size(rz,2)
  S = [S, sprintf('%3.2f   ',rz(i))];
end;
for i=1:2:size(cz,2)
    omega = sqrt(cz(i)^2 + cz(i+1)^2);
    chi = cz(i)/omega;
    S = [S, sprintf('(%3.2f %3.2f) [wn = %3.2f, chi = %3.2f]   ',cz(i),
cz(i+1), omega, chi)];
end;
S = [S, sprintf('\n  Poles: ')];
for i=1:size(pr,2)
  S = [S, sprintf('%3.2f   ',pr(i))];
end;
for i=1:2:size(pc,2)
    omega = sqrt(pc(i)^2 + pc(i+1)^2);
    chi = pc(i)/omega;
    S = [S, sprintf('(%3.2f %3.2f) [wn = %3.2f, chi = %3.2f]   ',pc(i),
pc(i+1), omega, chi)];
end;
disp(S);

result = S;
S = sprintf('  %s', m2);
for i=1:length(cost)
    S = [S, sprintf('%4.5f ', cost(i))];
end;
disp(S);

result = [result, S];

% ORDER4 Indexes weighting based on different criteria.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function [y, IM] = order4(X, a1, a2, a3, r1, r2, r3, nr1, nr2, nr3, nn);

[r, c] = size(X);
% Indexes weighting
for k=1:r
    dr1 = 0;
    dr2 = 0;
    dr3 = 0;
    if X(k,4) > 0  % Bounds violation
        dr1 = 1;
```

```
    end;
    if X(k,5) > 0  % Stability violation
        dr2 = 1;
    end;
    if X(k,6) > 0  % Lo with decreasing module violation
        dr3 = 1;
    end;
    if k > nn
        IM(k,:) = [X(k,1:3).*[a1 a2 a3], [dr1 dr2 dr3].*[r1 r2 r3].*...
            ([1/r1 1/r2 1/r3].*X(k,4:c) + 1)];
    else
        IM(k,:) = [X(k,1:3).*[a1 a2 a3], [dr1 dr2 dr3].*[nr1 nr2 nr3].*...
            ([1/nr1 1/nr2 1/nr3].*X(k,4:c) + 1)];
    end;
end;

MIV = sum(IM, 2); % Sum the elements of each row.
% To put in order and obtain the final order index.
[Ord2, III] = sort(MIV);

y = III;
for k=1:r
    y(III(k)) = k;
end;


%  RANDOMR Función that selects a random number, uniformly distributed,
%          within the interval.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function res = randomr(x, rang1, rang2, lambda)

% Total range
rang = (rang2 - rang1);

% Result limitation within the range
res = x + lambda*(rand(1) - 1)*rang;
if rang1 > res
    res = rang1;
end;
if rang2 < res
    res = rang2;
end;


% SELECTION Function to select the individuals of a population in terms of
%           their evaluation in the fitness function.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function [y, Iy, Cy]=selection(X,XCost,a1,a2,a3,r1,r2,r3,nr1,nr2,nr3,nn,n)

[FF, I] = order4(XCost, a1, a2, a3, r1, r2, r3, nr1, nr2, nr3,nn);
```

```
Iy = [];
y = [];
Cy = [];
for i=1:n
    pos = find(FF==i);
    y = [y; X(pos,:)];
    Iy = [Iy; I(pos,:)];
    Cy = [Cy; XCost(pos,:)];
end;


% STABLE Stability of the 1 + G(s)Po(s) characteristic equation. If any
%        root lies on the RHP the system is instable.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


function st = stable(kg,cz,rz,pc,pr,nP0,dP0)

% Transfer function for Lo(s)
[numG, denG] = TG(kg,cz,rz,pc,pr);
T2 = conv(numG, nP0);
T1 = conv(denG, dP0);

% Characteristic equation calculation
r1 = length(T1);
r2 = length(T2);
if (r1>r2)
    T2 = [zeros(1,r1-r2), T2];
else
    T1 = [zeros(1,r2-r1), T1];
end
T = T1 + T2;

% Characteristic equation roots
r=roots(T);
rreal = max(real(r));
if rreal > 0
    st = rreal;
else
    st = 0;
end

% TG Transfer function for G(s).The outcome is the numerator and the
%    denominator.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function [numG, denG] = TG(kg,cz,rz,pc,pr)

% ------ Transfer function for G(s) --------------
%                E (s/z + 1)
%   G(s) = kg ----------------
%                E (s/p + 1)
```

```
numG = kg;
denG = 1.0;

% Complex zeros
for j=1:2:size(cz,2)
    omega = cz(j)^2 + cz(j+1)^2;
    chi = cz(j);
    numG = conv(numG, [1/omega, 2*chi/omega, 1]);
end
% Real zeros
for j=1:size(rz,2)
    numG = conv(numG, [1/rz(j) 1]);
end

% Complex poles
for j=1:2:size(pc,2)
    omega = pc(j)^2 + pc(j+1)^2;
    chi = pc(j);
    denG = conv(denG, [1/omega, 2*chi/omega, 1]);
end

% Real poles
for j=1:size(pr,2)
    denG = conv(denG, [1/pr(j) 1]);
end


% TLo Transfer function for Lo(s) = G(s)Po(s)
%
%                     E (s + z)
%      Lo(s) = kg -------------- Po(s)
%                     E (s + p)
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------

function [mod, ph] = TLo(kg,cz,rz,pc,pr,nP0,dP0,freq)
[numG, denG] = TG(kg,cz,rz,pc,pr);
numL0 = conv(numG, nP0);
denL0 = conv(denG, dP0);

lmo = polyval(numL0, sqrt(-1)*freq) / polyval(denL0, sqrt(-1)*freq);
mod = abs(lmo);
ph = angle(lmo);

% -------- Phase between 0 and -2pi --------------------
if ph > 0
  ph = -2*pi + ph;
end;
if ph < -2*pi
  ph = ph + 2*pi*floor(-ph/(2*pi));
end;
ph = ph * 180 / pi;   % Convert the phase into degrees.

% TMPL-BNDS Function to calculate the problem templates and the bounds.
%
```

```
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-----------------------------------------------------------------

function [bdb, np, dp] = Tmpl_Bnds();

close all;
clc;
clear;

% ----------- Templates generation -----------------

disp('Plant templates calculation ...')
c=1;
for  k=[1 2.5 6 10]
    for a=[1 1.5 3 6 10]
        np(c,:)=[k*a];
        dp(c,:)=[1 a 0];
        cp(1,1,c)=tf(np(c,:),dp(c,:));
        c=c+1;
    end
end
disp('                                    ... Finished');

% ---------- Nominal plant parameters -----------

nompt = 1;    % Nominal plant is always in the first row.

% --------------- Frequency range --------------

w =[0.1 0.5 1 2 5 10 15 40 60 80 120 170 200 300 400 500 600 800 1000 ...
    1250 1500 1750 2000 3000 5000 10000];
plottmpl(w,cp,nompt);
set(gca,'Title',title('P(s) plant templates'));
pause(2);

% -----------------Bounds initialization--------------------------

W1=[];
W2=[];
W3=[];
W4=[];
W5=[];
W6=[];
W7=[];
W8=[];
W9=[];

% --------------- Bound Type 1 ------------------------

disp('Type 1 bounds calculation ...');
W1=1.2; save W1
wbd1=w;
bdb1 = sisobnds(1,wbd1,W1, cp);
plotbnds(bdb1);
set(gca,'Title',title('Stability bounds'));
disp('                                    ... Finished');
pause(2);
```

```
% -------------- Bound Type 7 ------------------------

wbdb7 = [0.1 0.5 1 10];
disp('Type 7 bounds calculation ...');
cmu = tf(0.6584*[1 30], [1 4 19.752]);
cml = tf(120, [1 17 82 120]);
W7 = [cmu; cml]; save W7
bdb7 = sisobnds(7, wbdb7, W7, cp);
plotbnds(bdb7);
set(gca,'Title',title('Tracking bounds'));
disp('                                    ... Finished');
pause(2);

% -------------- Bounds intersection ----------------

bdb = grpbnds(bdb1,bdb7);
bdb = sectbnds(bdb);
plotbnds(bdb);
set(gca,'Title',title('All bounds intersection'));
disp('Bounds calculation finished.');
disp(' ');
pause(2);


% UPH  Function necessary to determine how tangent Lo is to the UHFB
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-----------------------------------------------------------------------

function y = uph(bdb_a, bdb_b)

n = length(bdb_a(1,:));
m = length(bdb_a(:,1));
ph = 0:-5:-180;

for i=1:m
    bdba(i) = bdb_a(m - i + 1, n);
    bdbb(i) = bdb_b(m - i + 1, n);
end;

[bdb_aS, Ia] = sort(bdba(1:length(ph)));
[bdb_bS, Ib] = sort(bdbb(1:length(ph)));
bdb_aMin = bdb_aS(1);
bdb_bMax = bdb_bS(1);

for i=1:length(ph)
    Ph_a(i) = ph(Ia(i));
    Ph_b(i) = ph(Ib(i));
end;

bdb_aS2 = [];
bdb_bS2 = [];
Ph_a2 = [];
Ph_b2 = [];
for i=1:length(bdb_aS)
    if abs(bdb_aS(i)) ~= 320
```

```
        bdb_aS2 = [bdb_aS2, bdb_aS(i)];
        Ph_a2 = [Ph_a2, Ph_a(i)];
    end;
    if abs(bdb_bS(i)) ~= 320
        bdb_bS2 = [bdb_bS2, bdb_bS(i)];
        Ph_b2 = [Ph_b2, Ph_b(i)];
    end;
end;

bdb_u = [bdb_bS2, bdb_aS2];
Ph_u = [Ph_b2, Ph_a2];

y = [bdb_u; Ph_u];
```

8. APPENDIX B: MATLAB, C++ AND TEXT FILES FOR THE STRATEGY 2 (GENETIC ALGORITHMS)

8.1. MATLAB FILES

```
% AUTOLOOPSHAPING_AG  Program to sinthesys optimal controllers with an
%                     initial structure which can be modified by means of
%                     the optimal Hankel norm approximation. If any
%                     modification occurs the program sinthesys the optimal
%                     controller for thid new structure.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

% Genetic Algorithms

% First of all the user has to configure the CFG.TXT, CFGCONTR.TXT and
% CFGCOST.TXT files. Take into account that this program has been
% configured for controllers with up to five poles and zeros.


% In addition the plant and the performance specifications have to be
% defined, obtaining the plant templates and the bounds.

Tmpl_Bnds_AG; % Execute only if no template approximation is desired.
LOOPSHN; % Execute only for template approximations.

auxx='T';
auxx2=1;

while auxx=='T'

% Now the OPTQFT.exe is executed. As a result the following files are
% obtained: AAGG.LOG, PRIV1.TXT and VALIDVECTOR.TXT.

    !OPTQFT.exe

    [C,B,A,D] = textread('CFGCONTR.txt',...
        '%f %[^1,2,3,4,5] %f %[^1,2,3,4,5]','delimiter', ' ');

    B=2*A+1;

    aux2='T';

    % Once the user has the PRIV1.TXT the results are stored in the vectors
    % k(gains), zi(zeros), pi(poles), best_value(best fitness value) and
    % mean_value(generation fitness mean value).

    switch B

        case 11
            [best_value,mean_value,c1,k,z1,z2,z3,z4,z5,p1,p2,p3,p4,p5,c2]=...
                textread('priv1.txt','%f %f %c %f %f %f %f %f %f %f %f %f ...
                        %f %n %s','delimiter', ' ');
```

```
    [valids]=textread('validvector.txt','%u');
    % valids is a column vector with 1 for feasible solutions and
    % 0 for the non-feasible ones, between the generations of this
    % iteration.

    % From the above mentioned vectors only the feasible solution
    % are of interest, so the following lines extract them and
    % store in the vectors valid, gain, p and z.
    j=1;
    p=zeros(numel(find(valids(:)==1)),A);
    z=zeros(numel(find(valids(:)==1)),A);

    if numel(find(valids(:)==1))>0
        for i=1:length(valids)
            if valids(i)==1
                valid(j)=best_value(i);
                gain(j)=k(i);
                p(j,:)=[p1(i) p2(i) p3(i) p4(i) p5(i)];
                z(j,:)=[z1(i) z2(i) z3(i) z4(i) z5(i)];
                j=j+1;
            end
        end
    else
        aux2='F'; % No feasible solution exists.
    end


case 9
    [best_value,mean_value,c1,k,z1,z2,z3,z4,p1,p2,p3,p4,c2] = ...
        textread('priv1.txt','%f %f %c %f %f %f %f %f %f %f %f %n...
                %s','delimiter', ' ');
    [valids]=textread('validvector.txt','%u');

    j=1;
    p=zeros(numel(find(valids(:)==1)),A);
    z=zeros(numel(find(valids(:)==1)),A);

    if numel(find(valids(:)==1))>0
        for i=1:length(valids)
            if valids(i)==1
                valid(j)=best_value(i);
                gain(j)=k(i);
                p(j,:)=[p1(i) p2(i) p3(i) p4(i)];
                z(j,:)=[z1(i) z2(i) z3(i) z4(i)];
                j=j+1;
            end
        end
    else
        aux2='F'; % No feasible solution exists.
    end


case 7
    [best_value,mean_value,c1,k,z1,z2,z3,p1,p2,p3,c2] = ...
        textread('priv1.txt','%f %f %c %f %f %f %f %f %f %n %s',...
        'delimiter', ' ');
    [valids]=textread('validvector.txt','%u');
```

```
    j=1;
    p=zeros(numel(find(valids(:)==1)),A);
    z=zeros(numel(find(valids(:)==1)),A);

    if numel(find(valids(:)==1))>0
        for i=1:length(valids)
            if valids(i)==1
                valid(j)=best_value(i);
                gain(j)=k(i);
                p(j,:)=[p1(i) p2(i) p3(i)];
                z(j,:)=[z1(i) z2(i) z3(i)];
                j=j+1;
            end
        end
    else
        aux2='F'; % No feasible solution exists.
    end

case 5
    [best_value,mean_value,c1,k,z1,z2,p1,p2,c2] = ...
        textread('priv1.txt','%f %f %c %f %f %f %f %n %s',...
        'delimiter', ' ');
    [valids]=textread('validvector.txt','%u');

    j=1;
    p=zeros(numel(find(valids(:)==1)),A);
    z=zeros(numel(find(valids(:)==1)),A);

    if numel(find(valids(:)==1))>0
        for i=1:length(valids)
            if valids(i)==1
                valid(j)=best_value(i);
                gain(j)=k(i);
                p(j,:)=[p1(i) p2(i)];
                z(j,:)=[z1(i) z2(i)];
                j=j+1;
            end
        end
    else
        aux2='F'; % No feasible solution exists.
    end


case 3
    [best_value,mean_value,c1,k,z1,p1,c2] = ...
        textread('priv1.txt','%f %f %c %f %f %n %s','delimiter', ' ');
    [valids]=textread('validvector.txt','%u');

    j=1;
    p=zeros(numel(find(valids(:)==1)),A);
    z=zeros(numel(find(valids(:)==1)),A);

    if numel(find(valids(:)==1))>0
        for i=1:length(valids)
            if valids(i)==1
                valid(j)=best_value(i);
                gain(j)=k(i);
                p(j,:)=[p1(i)];
```

```
                    z(j,:)=[z1(i)];
                    j=j+1;
                end
            end
        else
            aux2='F'; % No feasible solution exists.
        end


end

m=A-1;

if aux2=='T' % If any feasible solution exists.

    I=find(min(valid(:))); % Best solution of each iteration.

    if auxx2==1
        zz=1; % Iterations counter. It is used to store the results.

        % Elements to store the results
        ac_gain=[];
        ac_p=zeros(9,A);% Number of possible controllers=9,max.number
                        % of poles=5
        ac_z=zeros(9,A);% Number of possible controllers=9,max.number
                        % of zeros=5
    end

    ac_gain(zz)=gain(I);
    for l=1:size(p,2)
        ac_p(zz,l)=p(I,l);
        ac_z(zz,l)=z(I,l);
    end

    % --------------------Model reduction---------------------
    %
    % Now the obtained controller is reduced to a m-th order model by
    % means of the optimal Hankel norm approximation.
    m=A-1;

    if m>=1
        sys=zpk(-ac_z(zz,:),-ac_p(zz,:),ac_gain(zz));
        sys=ss(sys);
        n=size(sys.A,1);
        nu=size(sys,2);
        [sysc,cinfo]=ncfmr(sys,n);
        sysch=hankelmr(cinfo.GL,m);
        syschm=minreal(inv(sysch(:,nu+1 ...
            :end))*sysch(:,1:nu));

        [Z,P,K]=zpkdata(syschm,'v'); % Gain, poles and zeros of the
                                     % reduced model.
        zz=zz+1;

        ac_gain(zz)=K;
        for l=1:size(P',2)
            ac_p(zz,l)=-P(l)';
            ac_z(zz,l)=-Z(l)';
```

```
            end
            zz=zz+1;

        end

    end


    if m>=1

        clear k p1 p2 p3 p4 p5 z1 z2 z3 z4 z5 p z valids valid gain ...
            best_value mean_value;
        delete priv1.txt
        delete aagg.log
        delete validvector.txt

        % Update the CFG.txt y CFGCONTR.txt. The original files are lost.

        File_Update;

    elseif m<1

        auxx='F';
        disp('controllers calculation finished')
        for h=1:numel(find(ac_p(:,1)~=0))
            ac_g=[];
            ac_poles=[];
            ac_zeros=[];
            for l=1:numel(find(ac_p(h,:)~=0))
                ac_g(h)=ac_gain(h);
                ac_poles(l)=-ac_p(h,l);
                ac_zeros(l)=-ac_z(h,l);
            end
            G(h)=zpk(ac_zeros(:),ac_poles(:),ac_g(h));
            clear ac_g ac_poles ac_zeros;
        end
        menuplot;
    end
    auxx2=auxx2+1;
end


% B2AAGG Function to generate a file containing the bounds variable, in an
%        understable format for the optqft program.
%
%        The bounds calculated by the QFT matlab toolbox has the following
%        format:
%
%       - each column corresponds to a different frequency
%       - if there are n analysed points, there are 2n+2 rows
%       - by default there are 73 phase points ([0:-5:-360])
%       - the first n values correspond to the minimum limit
%         (solid line)
%       - the next n values correspond to the maximum limit
%         (dashed line)
%       - the penultimate value corresponds to the frequency
%       - the last value indicates the type of bound
%
```

```
%       The output format is:
%       - each row corresponds to a different frequency
%       - if there are n phase analysed points, there are 2n+2 columns
%       - the first value corresponds to the frequency
%       - the next value corresponds to the number of points (73 by default)
%       - the next n values correspond to the minimum limit
%       - the next n values correspond to the maximum limit

% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function [] = b2aagg (bounds, Filename)


if ( ~isstr(Filename) ) error ('second parameter: file name'); end

[rows, columns] = size (bounds);
numPhases = (rows-2)/2;
b=bounds(1:rows-2,:);
b=[bounds(rows-1,:); numPhases*ones(1,columns); b];
b=b';

% save nombreFic b -ascii
eval(['save ' Filename ' b -ascii']);



% CHANGLE  Internal utility of convexhl
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function ang=changle(point1, point2)

dx=real(point2)-real(point1);
dy=imag(point2)-imag(point1);
ang=atan2(dy,dx);
if(ang<0) ang=ang+360; end



% CHDIST  Internal utility of convexhl. Calculation of the square of the
%         distance.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


function dist=chdist(point1, point2)

dx=real(point2)-real(point1);
dy=imag(point2)-imag(point1);
dist = dx*dx + dy*dy;
```

```
% CHREMOVE  Internal utility of convexhl. It removes the duplicated points.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


function v=chremove(complexvector)

v=[];
[rows, columns]=size(complexvector);
for i=1:rows
   [newrows, columns]=size(v);
   repeated = 0;
   for j=1:newrows
      if (equals(real(complexvector(i)),real(v(j))) && ...
          equals(imag(complexvector(i)),imag(v(j))) )
         repeated = 1;
         break;
      end
   end
   if (~repeated)
       v=[v; complexvector(i)];
   end
end


% CONVEXHL  Function that gives the convex-hull of a set of complex points
%           , but calculated in the db-phase plane.
%
%           complexvector: Nx1 of complex points vector.
%
%           convex: The M points of the convex polygon that contains all
%                   the points arranged anticlockwise.
%
%           The M point matches with the 1 in order to close the polygon.
%

% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


function convex=convexhl(complexvector)

[rows, columns]=size(complexvector);
if (columns > 1) error ('it has to be a Nx1 vector'); end
if (rows < 3) error ('At least 3 points are needed'); end

vector=chremove(complexvector);  % quitar repetidos
[rows, columns]=size(vector);
if (rows < 3) error ('At least 3 different points are needed'); end

% Mantain two complex vectors:the real one and the other one composed of
% (phase + i * db)
vectorReal = vector;
[db,phase]=dbphase(vector);
```

```
vector = phase + i * db;
% The calculations are made in "vector", but the final points are stored in
% "vectorReal"
%
% Algorithm:
%
%   1. Find the point with the lowest "y"
%   2. Exchange between the first element and the lowest one.
%      The first position is i=1 point of the convex-hull.
%   3. Find the i=i+1 of the convex-hull:
%      measure the angle between the i point and the rest (with respect to
%      the x axes). Take into account that the convex-hull angles are
%      always increasing. The new point is the one with the lowest angle
%      between all the points that have higher angle than the i point.
%

% Calculate the minimum y value
imin=1;
for ii=2:rows
   if(imag(vector(ii))<imag(vector(imin))) imin=ii;
   elseif (imag(vector(ii)) == imag(vector(imin)))
      % Take the closest one to the left
      if(real(vector(ii))<real(vector(imin))) imin=ii; end
   end
end

% swap
tmp=vector(1);  vector(1)=vector(imin);  vector(imin)=tmp;
tmp=vectorReal(1);  vectorReal(1)=vectorReal(imin);  vectorReal(imin)=tmp;

currentangle = 0;  % Angle of the i-th element, the i+1-th element must be
                   % higher.

% For each element i of the convex-hull
for ii=1:rows
   % Compare with the remainder looking for the one with minimum angle.
   if (ii==1)
      jmin=2;
      angMin = changle(vector(1), vector(2));
      distMin = chdist(vector(1), vector(2));
   else
      jmin=1;
      angMin = changle(vector(ii), vector(1));
      distMin = chdist(vector(ii), vector(1));
   end

   for j=ii+1:rows
      ang = changle(vector(ii), vector(j));
      dist = chdist (vector(ii), vector(j));
      % Find the minimum angle between all that beats the current.
      if (ang >= currentangle)
         if(ang<angMin)
            jmin=j;
            angMin = ang;
            distMin = dist;
         elseif (ang == angMin)
            % Take the closest one
            if (dist<distMin)
               jmin=j;
```

```
                angMin = ang;
                distMin = dist;
            end
        end
    end

end  % for j

% If the closest one is 1, end
if (jmin==1)
    convex=vectorReal(1:ii);
    convex=[convex; convex(1)];    % close the polygon
    return
else
    % swap
    tmp=vector(ii+1);  vector(ii+1)=vector(jmin);  vector(jmin)=tmp;
    tmp=vectorReal(ii+1);  vectorReal(ii+1)=vectorReal(jmin);
    vectorReal(jmin)=tmp;
end

currentangle = angMin;

end



% DBPHASE  Conversion from the complex plane to the dB-phase plane.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


function [db, phase] = dbphase (complexmatrix)

db=20*log10(abs(complexmatrix));
phase=180*atan2(imag(complexmatrix),real(complexmatrix))/pi;



% EQUALS  Function that returns 1 if v1 and v2 are approximately equal.
%
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


function isequal=equals(v1, v2)

delta=0.0000001;
isequal = ( (v1<v2+delta) & (v1>v2-delta) );
```

```
% FILE_UPDATE  Function to update the CFG.txt and CFGCONTR.txt files during
%              the execution of the Autoloopshaping_AG.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


function File_Update()
fid = fopen('CFG.TXT','r');

fout = fopen('out1.txt','wt');

a = textscan(fid, '%d %s %s %s%*[^\n]', 1);
n = a{1}-2;
fprintf(fout, '%d %s %s %s\n',n,a{2}{1},a{3}{1},a{4}{1});
for i = 1:a{1}
    b = textscan(fid, '%n %s %s %s %s %s %*[^\n]', 1);
    if (i<=n)
        fprintf(fout, '%d %s %s %s %s %s \n',b{1},b{2}{1},b{3}{1},...
            b{4}{1},b{5}{1},b{6}{1});
    end
end

for i = 1:a{1}
    b = textscan(fid, '%n %s %s %s %s %s %*[^\n]', 1);
    if (i<=n)
        fprintf(fout, '%d %s %s %s %s %s \n',b{1},b{2}{1},b{3}{1},...
            b{4}{1},b{5}{1},b{6}{1});
    end
end

% Parameters Precision
textLine = textscan(fid, '%n %s %s %s %s %s %*[^\n]', 1);
fprintf(fout, '%d %s %s %s %s %s \n',textLine{1},textLine{2}{1},...
      textLine{3}{1},textLine{4}{1},textLine{5}{1},textLine{6}{1});

% Number of Individuals in the population
textLine = textscan(fid, '%d %s %*[^\n]', 1);
fprintf(fout, '%d %s \n',textLine{1},textLine{2}{1});

% Crossover probability
textLine = textscan(fid, '%n %s %s %s %*[^\n]', 1);
fprintf(fout, '%d %s %s %s \n',textLine{1},textLine{2}{1},...
      textLine{3}{1},textLine{4}{1});

% Mutation probability
textLine = textscan(fid, '%n %s %s %s %*[^\n]', 1);
fprintf(fout, '%d %s %s %s \n',textLine{1},textLine{2}{1},...
      textLine{3}{1},textLine{4}{1});

% Show continuously (1) or just at the end (0)
textLine = textscan(fid, '%d %s %s %s %s %s %s %s %s %*[^\n]', 1);
fprintf(fout, '%d %s %s %s %s %s %s %s %s \n',textLine{1},textLine{2}{1},...
      textLine{3}{1},textLine{4}{1},textLine{5}{1},textLine{6}{1},...
      textLine{7}{1},textLine{8}{1},textLine{9}{1});
```

```
% Maximum number of generations
textLine = textscan(fid, '%d %s %s %s %s %*[^\n]', 1);
fprintf(fout, '%d %s %s %s %s \n',textLine{1},textLine{2}{1},...
      textLine{3}{1},textLine{4}{1},textLine{5}{1});

% Minimum desired fitness
textLine = textscan(fid, '%d %s %s %s%*[^\n]', 1);
fprintf(fout, '%d %s %s %s \n',textLine{1},textLine{2}{1},...
      textLine{3}{1},textLine{4}{1});

fclose(fout);
fclose(fid);

fid = fopen('CFGCONTR.TXT','r');
fout = fopen('out2.txt','wt');

%Number of controller fixed poles
a = textscan(fid, '%d %s %s %s %s %s %s%*[^\n]', 1);
fprintf(fout, '%d %s %s %s %s %s %s\n',a{1},a{2}{1},a{3}{1},...
      a{4}{1},a{5}{1},a{6}{1},a{7}{1});

%Number of parameters that correspond to the controller zeros
a = textscan(fid, '%d %s %s %s %s %s %s %s %s %s%*[^\n]', 1);
n = a{1}-1;
fprintf(fout,'%d %s %s %s %s %s %s %s %s %s %s \n',n,a{2}{1},a{3}{1},...
      a{4}{1},a{5}{1},a{6}{1},a{7}{1},a{8}{1},a{9}{1},a{10}{1});

fclose(fout);
fclose(fid);

movefile out1.txt CFG.TXT
movefile out2.txt CFGCONTR.TXT

end



% FRECSN  Function to define the frequency vector.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function w=frecsn()

w=[0.01 0.05 0.1 0.2 1.0 5.0 10 20 30 40 50 100];



% GRIDQFTN Function to determine the number of points of each plant
%          parameter (K, Z, P, W, D) that are considered.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------
```

```
function nump=gridqftn()

nump = 3;



% LIMITSN  Function that defines the range of variation of each plant
%          parameter.
%
%           Plant = K * (1+s/z) / s (1+s/p) (1+2Ds/W+s^2/W^2)
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function [Kmin,Kmax,Zmin,Zmax,Pmin,Pmax,Wmin,Wmax,Dmin,Dmax]=limitsn()

Kmin = 0.2;
Kmax = 2;
Zmin = 0.5;
Zmax = 0.75;
Pmin = 1;
Pmax = 10;
Wmin = 5;
Wmax = 6;
Dmin = 0.8;
Dmax = 0.9;



% LOOPSHN  Plant and performance specifications definition and plant
%          templates and QFT bounds creation. Use when rectangular or
%          convex-hull approximation is desired.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


% Nominal Plant Parameters Values
Knom = 2;
Znom = 0.5;
Pnom = 10;
Wnom = 6;
Dnom = 0.8;

% Frequencies to consider
w=frecsn;
[m,numFrecs]=size(w);

% Templates for the nominal plant
ptemplates=tmplnx(Knom, Znom, Pnom, Wnom, Dnom);

% Nominal Plant
nompt = 1;
```

```
%% Plant templates %%
plottmpl(w,ptemplates,nompt);
%% Bounds calculation %%

W1=[];
W2=[];
W3=[];
W4=[];
W5=[];
W6=[];
W7=[];
W8=[];
W9=[];

R=0; %  Delay

% Bound type 1: u-contour in 6 dB (robust stability)
w1 = w;
W1 = 10^(6/20);     % 20 log(x) = 6
bound1 = sisobnds(1,w1,W1,ptemplates,R,nompt);
plotbnds(bound1);
title('Bound type 1');
save W1

% Bound type 7: Input reference tracking
w7=[0.01 0.05 0.1 0.2 1.0 5.0 10.0];
numA = 1;
denA = conv ([1 1], conv ([1 1], [1/2 1]));
inf = tf(numA,denA);
numB = [1/0.35 1];
denB = conv ([1/0.5 1], [1/3 1]);
sup = tf(numB,denB);
W7 = [sup; inf];
bound7 = sisobnds(7,w7,W7,ptemplates,R,nompt);
plotbnds(bound7);
title('Bound type 7');
save W7

% Group bounds
bounds=grpbnds(bound1,bound7);

% Bounds intersection
bounds=sectbnds(bounds);
plotbnds(bounds);
title('Bounds Intersection');

% 'bnd_n' file generation for the Genetic Algorithm
B2AAGG (bounds, 'bnd_n')

% Frequency array for the loop-shaping
wloop = logspace(-3,3,200);
save wloop
% Frequency array for the chksiso
wcheck = logspace(-3,3,300);
save wcheck

% Nominal Plant
nump = Knom * [1/Znom 1];
```

```
denp = conv ( [1 0], conv ([1/Pnom 1], [1/Wnom^2 2*Dnom/Wnom 1]) );
cp=tf(nump,denp);



% MENUPLOT Menu to choose the task to do with the results obtained from the
%          automatic loop-shaping.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

k=menu('Choose an option','step','bode','nichols','nyquist','chksiso');

switch k
    case 1
        for h=1:numel(find(ac_p(:,1)~=0))
            figure(2)
            step(cp*G(h));
            grid on;
            set(gca, 'FontSize',[5])
            axis([0 70 0 200])
            hold all
        end
    case 2
        for h=1:numel(find(ac_p(:,1)~=0))
            figure(2)
            bode(cp*G(h),w);
            grid on;
            set(gca, 'FontSize',[5])
            axis([0.001 10000 -180 -45])
            hold all
        end
    case 3
        load wloop;
        for h=1:numel(find(ac_p(:,1)~=0))
            lpshape(wloop, bounds, cp, G(h), []);
        end
    case 4
        for h=1:numel(find(ac_p(:,1)~=0))
            figure(2)
            nyquist(cp*G(h));
            grid on;
            set(gca,'FontSize',[5])
            axis([-6 6 -3 3])
            hold all
        end
    case 5
        load wcheck;
        load W1
        load W7
        for h=1:numel(find(ac_p(:,1)~=0))
            if ~isempty(W1)
            chksiso(1,wcheck,W1,cp,0,G(h),1,1);
            title(['bound type 1 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W2)
            chksiso(2,wcheck,W2,cp,0,G(h),1,1);
            title(['bound type 2 with the ',num2str(h),'th controller']);
```

```
            grid on;
            end
            if ~isempty(W3)
            chksiso(3,wcheck,W3,cp,0,G(h),1,1);
            title(['bound type 3 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W4)
            chksiso(4,wcheck,W4,cp,0,G(h),1,1);
            title(['bound type 4 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W5)
            chksiso(5,wcheck,W5,cp,0,G(h),1,1);
            title(['bound type 5 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W6)
            chksiso(6,wcheck,W6,cp,0,G(h),1,1);
            title(['bound type 6 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W7)
            chksiso(7,wcheck,W7,cp,0,G(h),1,1);
            title(['bound type 7 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W8)
            chksiso(8,wcheck,W8,cp,0,G(h),1,1);
            title(['bound type 8 with the ',num2str(h),'th controller']);
            grid on;
            end
            if ~isempty(W9)
            chksiso(9,wcheck,W9,cp,0,G(h),1,1);
            title(['bound type 9 with the ',num2str(h),'th controller']);
            grid on;
            end
        end
end
```

```
% PLANTN  Function to calculate the plant equation.
%
% Planta = K * (1+s/z) / s (1+s/p) (1+2Ds/W+s^2/W^2)
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function g=plantn(K, Z, P, W, D, w)

% w is a row vector

g = K .* (1+j*w/Z) ./ ( (j*w) .* (1+j*w/P) .* (1+2*D*j*w/W-w.*w/W^2) );
```

```
% RECTANG  Function that makes the rectangular approximation of the
%          templates(the smaller rectangle that contains the original
%          template).
%
%          Each column is a templateand each row indicates a new point.
%          NumPointSide indicates the number of points that will appear in
%          side of the rectangle.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function tmplRect=rectang(templates,numPointSide)

[numVertexes, numTemplates]=size(templates);
[db,phase]=dbphase(templates);

for c=1:numTemplates

   mindb=100000;   % valor mínimo en dB
   down=0;         % índice al punto que tiene mindb
   downphase=0;    % fase del punto "abajo"
   maxdb=-100000;  % valor máximo en dB
   up=0;           % índice al punto que tiene maxdb
   upphase=0;      % fase del punto "arriba"
   minphase=10000; % valor mínimo en fase
   left=0;         % índice al punto que tiene minfase
   dbleft=0;       % dB del punto "izq"
   maxphase=-10000;% valor máximo en fase
   right=0;        % índice al punto que tiene maxfase
   dbright=0;      % dB del punto "der"

   for f=1:numVertexes
      [db, phase] = dbphase(templates(f,c));
      if (db > maxdb) maxdb = db; up = f; upphase = phase; end
      if (db < mindb) mindb = db; down = f; downphase = phase; end
      if (phase > maxphase) maxphase = phase; right = f; dbright = db; end
      if (phase < minphase) minphase = phase; left = f; dbleft = db; end
   end

   % "right" point downwards displacement till it has the same db value as
   % the "down" point.
   displInDb = mindb - dbright;
   bottomrightpoint = templates(right,c) * 10^(displInDb/20);

   % "up" point displacement to the right till it has the same phase value
   % as  the "right" point.
   displInPhase = (maxphase - upphase)*pi/180;
   toprightpoint = templates(up,c) * exp(j*displInPhase);

   % "left" point upwards displacement till it has the same db value as
   % the "up" point.
   displInDb = maxdb - dbleft;
   topleftpoint = templates(left,c) * 10^(displInDb/20);

   % "down" point displacement to the left till it has the same phase value
   % as  the "left" point.
   displInPhase = (minphase - downphase)*pi/180;
```

```
   bottomleftpoint = templates(down,c) * exp(j*displInPhase);

   % Displacement of every final point with respect to the previous one.
   if (numPointSide < 2) numPointSide = 2; end
   deltaInDb = (maxdb-mindb)/(numPointSide-1);
   deltaInPhase = (maxphase-minphase)/(numPointSide-1);
   deltaInPhase = deltaInPhase*pi/180;

   % Finally, all the points.
   for f=1:numPointSide
      dispD=deltaInDb*(f-1);
      dispPh=deltaInPhase*(f-1);
      tmplRect(f,c)=bottomrightpoint * 10^(dispD/20);
      tmplRect(numPointSide+f,c)=toprightpoint * exp(-j*dispPh);
      tmplRect(2*numPointSide+f,c)=topleftpoint * 10^(-dispD/20);
      tmplRect(3*numPointSide+f,c)=bottomleftpoint * exp(j*dispPh);
   end

end



% TMPL_BNDS_AG  Plant and performance specifications definition and plant
%               templates and QFT bounds creation. Use when no templates
%               approximation is desired.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

% In line 32 of menuplot.m write "cp(1,1,nompt)" where "cp".If not it will
% not work

% Plant definition
c=1;
for K = linspace(0.2,2,3)
    for Z = linspace(0.5,0.75,3)
        for P = linspace(1,10,3)
            for Wn = linspace(5,6,3)
                for D = linspace(0.8,0.9,3)
                    np(c,:) = K*[1/Z 1];
                    dp(c,:) = conv([1 0],conv([1/P 1],[1/Wn^2 2*D/Wn 1]) );
                    cp(1,1,c)=tf(np(c,:),dp(c,:));
                    c=c+1;
                end
            end
        end
    end
end

clear K Z P Wn D c;

% Nominal plant
nompt = 187;

% Frequencies of interest
w=frecsn;
```

```
%% Templates calculation %%

plottmpl(w,cp,nompt);

%% Bounds calculation %%

W1=[];
W2=[];
W3=[];
W4=[];
W5=[];
W6=[];
W7=[];
W8=[];
W9=[];

R=0; % Delay

% Bound type 1: u-contour in 6 dB (robust stability)
w1 = w;
W1 = 10^(6/20);     % 20 log(x) = 6
bound1 = sisobnds(1,w1,W1,cp,R,nompt);
plotbnds(bound1);
title('Bound type 1');
save W1

% Bound type 7: Input reference tracking
w7=[0.01 0.05 0.1 0.2 1.0 5.0 10.0];
numA = 1;
denA = conv ([1 1], conv ([1 1], [1/2 1]));
inf = tf(numA,denA);
numB = [1/0.35 1];
denB = conv ([1/0.5 1], [1/3 1]);
sup = tf(numB,denB);
W7 = [sup; inf];
bound7 = sisobnds(7,w7,W7,cp,R,nompt);
plotbnds(bound7);
title('Bound type 7');
save W7

% Group bounds
bounds=grpbnds(bound1,bound7);

% Bounds Intersection
bounds=sectbnds(bounds);
plotbnds(bounds);
title('Bounds Intersection');

% 'bnd_n' file generation for the Genetic Algorithm
B2AAGG (bounds, 'bnd_n')

% Frequency array for the loop-shaping
wloop = logspace(-3,3,200);
save wloop

% Frequency array for the chksiso
wcheck = logspace(-3,3,300);
```

```
save wcheck
% TMPLN  Function to calculate the plant templates.
%        Plant = K * (1+s/z) / s (1+s/p) (1+2Ds/W+s^2/W^2)
%
%        w is a row vector.
%        Note that the template for w=10 has a discontinuity in phase and
%        is badly calculated. Due to this, the points are displaced.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function templates=tmpln()

[Kmin,Kmax,Zmin,Zmax,Pmin,Pmax,Wmin,Wmax,Dmin,Dmax]=limitsn;
nump=gridqftn;
w=frecsn;

complexmatrix = [];

% Generate the points
for K = vector(Kmin, Kmax ,nump)
   for Z = vector(Zmin, Zmax, nump)
      for P = vector(Pmin, Pmax, nump)
         for W = vector(Wmin, Wmax, nump)
            for D = vector(Dmin, Dmax, nump)
               line = plantn (K,Z,P,W,D,w);
               complexmatrix = [complexmatrix; line];
            end
         end
      end
   end
end

[mudo, numTemplates]=size(w);

% Displace w=10
complexmatrix(:,find(w==10))=complexmatrix(:,find(w==10)).* exp(-j*pi*3/2);

templates=[];
for i=1:numTemplates
   convex=convexhl(complexmatrix(:,i));
   % There may be dimension problems if the number of point of each
   % convex-hull does not match.
   [numVertexesAnt, mudo]=size(templates);
   [numVertexes, mudo]=size(convex);
   if (numVertexesAnt>0)
      dif = numVertexes - numVertexesAnt;
      if (dif > 0)
         % Complete "templates"
         for i=1:dif  templates=[templates; templates(numVertexesAnt,:)]; end
      elseif (dif < 0)
         % Complete "convex"
         for i=1:-dif  convex=[convex; convex(numVertexes,:)]; end
      end
   end

   templates = [templates convex];
```

```
end
% TMPLNX  Function to calculate the Thompson-Nwokah plant templates, adding
%         the nominal plant that is used in QFT. This function is similar
%         to TMPLN with the difference that the (Knom,Znom,Pnom,Wnom,Dnom)
%         plant is added as the first of the templates.
%
%         * TMPLN is used to calculate the area
%         * TMPLNX is used as template in QFT (nomp = 1)
%
%         w is a row vector.
%
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------

function tmpl=tmplnx(Knom, Znom, Pnom, Wnom, Dnom)

% The first plant of the template
w=frecsn;
tmpl = plantn ( Knom, Znom, Pnom, Wnom, Dnom, w );

% Displacement of w=10
tmpl(:,find(w==10))=tmpl(:,find(w==10)) .* exp (-j*pi*3/2);

% The remainder
tmp=tmpln;

% Thompson-Nwokah uses a rectangular approximation of the templates.
tmp=rectang(tmp,10);

% Remove the line 37 if no rectangular approximation is desired.

% The total
tmpl = [tmpl; tmp];



% VECTOR   Function that gives vector with the data logarithmically spaced
%          out. vmin is the initial value, vmax is the final value and nump
%          is the number of values of the vector.
%
% Authors: Mario Garcia-Sanz, Carlos Molins
% 1.September.2009
%-------------------------------------------------------------------------


function v=vector(vmin,vmax,nump)

v = logspace(log10(vmin),log10(vmax),nump);
```

8.2. C++ FILES

```
// AAGG.h   Classes definition for AAGG.cpp.
//
// Authors: Mario Garcia-Sanz, Carlos Molins
// 1.September.2009
//-----------------------------------------------------------------------

#if !defined __AAGG_H__
#define __AAGG_H__

#include <vector>
#include <fstream>


// Class that has the structure of each solution (genotype)
// A Chromosome is a set of booleans (genes)
// (The genes are grouped to form parameters)
//
// Index of the first gene = 0
//

class Chromosome
{
   private:

      bool * genes;
      int numTotalGenes;

      // changes one gene
      void change (int indGen);

   public:

      Chromosome ( int numG );
      Chromosome ( const Chromosome & father, const Chromosome & mother,
              int Crosspoint );
      // Crosspoint = 0 --> first gene of the father, the rest of the
        // mother
      // Crosspoint = numTotalGenes-1 --> all the genes of the father
      ~Chromosome (){ delete [] genes; }

      Chromosome ( const Chromosome & );
      Chromosome & operator = ( const Chromosome & );

      // gives the integer value represented by a set of genes,
        // supposing that the one with the lowest index is the less
        // significative
      int tovalue (int indGenInitial, int numGenes) const;

      // mutates genes with a certain probability
      void mutate ();

};



// Represents a set of genes that forms a unity

class Parameter {

   private:
```

```
      int numGenes;
      double valueMin;   // minimum parameter value permitted
      double valueMax;   // maximum parameter value permitted
      int valueDecimalMax;  // 2**numGenes - 1

   public:

      Parameter (){}  // necessary to use later std::vector
      Parameter ( int numG, double vMin, double vMax );

      // By default:
      // Parameter ( const Parameter & );
      // Parameter & operator = ( const Parameter & );

      int getNumGenes() const { return numGenes; }
      // given a set of genes, it translates to a parameter range
      double tovalue ( int valueDecimal ) const;

      // necessary to use later std::vector
      bool operator< ( const Parameter & p ) const { return true; }
      bool operator== ( const Parameter & p ) const { return false; }
};



// Set of parameters, shared by all the individuals:

class Parameters {

      static std::vector <Parameter> Parameterslist;

   public:

      static void addParameter ( int numGenes, double valueMin,
              double valueMax );
      static const Parameter & getParameter (int i){
              return Parameterslist[i]; }

      static int getNumParameters (){
              return Parameterslist.size(); }
      static int getNumTotalGenes ();
};



// Phenotype: coded values by the genotype (saved in the genes
// structure): one value for each parameter

class Phenotype {

   private:

      double * values;

   public:

      Phenotype ( const Chromosome & );
```

```
      ~Phenotype (){ delete [] values; }

      Phenotype ( const Phenotype & );
      Phenotype & operator = ( const Phenotype & );

        // index = 0 to obtain the value of the first parameter
      double getParameterValue (int index) const {
             return values [index]; }
};


// Utility class to implement the scale change of the fitness
// function
//
// The fitness function f has two problems:
//  * At the beginning the mediocre individuals disappear,
//    because they are clearly worse than others. Useful
//    genetic information is lost (what is called the
//    "premature convergence").
//  * At the end almos all the individuals are similar.
//    The really best hardly stand out ("random walk" effect)
//
// The solution is a new fitness function F=a*f+b, that
//  - An average individual of f has, on average, one offspring.
//    F must mantain this. That is to say, faverage==Faverage
//  - We imposse that the best individual of F has, on average,
//    K offsprings, i.e., Fmax==K*Faverage
//

class Scale {

   private:

      static const int K;
      static double a;
      static double b;

   public:

      // calculates the new a,b given certain data of the
        //current population
      static void recalculateScale ( double fitnessMin,
             double fitnessMax, double fitnessAvg );
      // evaluate the new fitness F
      static double getFitnessScaled ( double fitness );
};



// Represents a possible solution to the problem

class Individual {

   private:

      Chromosome * genotype;
        Phenotype * phenotype;
      double fitnessReal;              // the real
      mutable double fitnessofWork;  // F=a*f+b
      // the fitnessofWork is unknown at the birth of the
```

```
        // individual.
      // It os known when all the generation is formed.

      void evaluate ();  // calculate the real fitness

   public:

      Individual ();
      Individual ( const Individual & father,
              const Individual & mother, int Crosspoint );
      // Crosspoint = 0 --> first gene of the father, the
        // rest of the mother
      // Crosspoint = numTotalGenes-1 --> all the genes of
        // the father
      ~Individual ();

      Individual ( const Individual & );
      Individual & operator = ( const Individual & );

      void mutate ();

      double getFitnessReal () const { return fitnessReal; }
      double getFitnessofWork () const {
              return fitnessofWork; }
      void updateFitnessofWork () const { fitnessofWork
              = Scale::getFitnessScaled ( fitnessReal ); }
      bool isValid () const;  // violates or not the possible
                                          // constraints

      // to put in order from lower to higher, with the best
        // at the beginning
      bool operator< ( const Individual & ind ) const { return
              getFitnessReal() > ind.getFitnessReal(); }
      bool operator== ( const Individual & ind ) const { return
              getFitnessReal() == ind.getFitnessReal(); }
      friend std::ostream & operator<< (std::ostream & os, const
              Individual &);
};


std::ostream & operator<< (std::ostream & os, const Individual
                                        & indiv);


// Utility class to implement the selection by means of the
// "stochastic remainder"

class SRemainder {

   private:

      // each element of the array is an index between 0 and
        //numIndividuals-1
      int * arrayIndexes;
      int numIndividuals;
      int unSelected;  // number of individuals that are to
                       // be selected
      SRemainder ( const SRemainder & );
      SRemainder & operator = ( const SRemainder & );
```

```
   public:

      SRemainder ( const std::vector <Individual> &
              Individualslist, double avgFitnessofWork );
      ~SRemainder ();

      int getIndex ();
};


// Represents the set of individuals put in order by their
// fitness value (from best to worst)

class Population {

   private:

      std::vector <Individual> Individualslist;
      double sumFitnessReal; // sum fitnessReal of
                                          // each individual
      double maxFitnessReal; // maximum fitnessReal value
                                          // that is in the population
      double minFitnessReal; // minimum fitnessReal value
                                          // that is in the population
      double sumFitnessofWork; // sum fitnessofWork of
                                           // each individual
      mutable SRemainder * sremainder;

      Population ( const Population & );
      Population & operator = ( const Population & );

   public:

      Population ();                  // empty population
      Population ( int numIndividuals );// random population
      ~Population ();

      void addIndividual ( const Individual & );
      // calculates the fitnessofWork
      void endofaGeneration ();

      const Individual & select () const;
      double getFitnessRealAverage () const { return
              sumFitnessReal / (double) Individualslist.size(); }
      double getMaxFitnessReal () const { return
              maxFitnessReal; }
      double getMinFitnessReal () const { return
              minFitnessReal; }
      double getBestFitnessReal () const { return
              Individualslist[0].getFitnessReal(); }
      double getFitnessofWorkAverage () const { return
              sumFitnessofWork / (double) Individualslist.size(); }
      const Individual & getBestIndividual () const { return
              Individualslist[0]; }
};


std::ostream & operator<< (std::ostream & os, const
                                    Population & pob);
```

```
// Utility class that represents the Genetic Algorithm itself

class GeneticAlgorithm {

   private:

      Population * thePopulation;
      Population * theNewPopulation;   // next generation
      int numGenerations;
      bool showContinuous;
      int numGenerationsMax;
      double minFitnessRealDesired;

      GeneticAlgorithm ( const GeneticAlgorithm & );
      GeneticAlgorithm & operator = ( const GeneticAlgorithm & );

      const Individual & select ();
      void cross (const Individual & father, const
              Individual & mother);
      void mutate (Individual & indiv);
      void passGeneration ();
      bool stop ();       // indicates the end of the algorithm.
      void show ( std::ostream & os );

   public:

      GeneticAlgorithm ();
      ~GeneticAlgorithm ();

      void initialize ( int numIndividuals, double pCross, double
              pmutate, bool showContinuous, int maxGenerations,
              double minFitnessReal );
      void mainAlgorithm ( std::ostream & os );

};
#endif


//AAGG.cpp  Basic Genetic Algorithm.C++ adaptation of the
//                Goldberg basic code [23].
//
//
// Authors: Mario Garcia-Sanz, Carlos Molins
// 1.September.2009
//---------------------------------------------------------------------

#include <math.h>          // for floor()
#include <time.h>          // for time()
#include <algorithm>       // for sort()
#include <float.h>         // for DBL_MAX
#include "random.h"
#include "aagg.h"
// Problem dependent parameters
int numIndividualsInPopulation;
double pMutate;
double pCross;
```

```
// External evaluation function
extern double externalEvaluation ( const Phenotype & );
// Function to check the external validity
// (no constraint violation)
extern bool isValidExternalSolution ( const Phenotype & );



Chromosome::Chromosome ( int numG ){
   genes = new bool [numTotalGenes = numG ];
   // random initialization
   for ( int i = 0; i < numG; i++ ) genes [i] = flip (0.5);
}

Chromosome::Chromosome ( const Chromosome & father, const Chromosome &
                                    mother, int Crosspoint ){
   // Crosspoint = 0 --> father´s first gene, the rest of the mother
   // Crosspoint = numTotalGenes-1 --> all the gens of the father
   genes = new bool [ numTotalGenes = father.numTotalGenes ];
   for ( int i = 0; i <= Crosspoint; i++ ) genes [i] = father.genes[i];
   for ( int i = Crosspoint + 1; i < numTotalGenes; i++ ) genes [i] =
         mother.genes[i];
}

Chromosome::Chromosome ( const Chromosome & crom ){
   numTotalGenes = crom.numTotalGenes;
   genes = new bool [numTotalGenes ];
   for (int i = 0; i < numTotalGenes; i++) genes[i] = crom.genes[i];
}

Chromosome & Chromosome::operator = ( const Chromosome & crom ){
   if ( &crom != this ){
      delete [] genes;
      numTotalGenes = crom.numTotalGenes;
      genes = new bool [numTotalGenes ];
      for (int i = 0; i < numTotalGenes; i++) genes[i] = crom.genes[i];
   }
   return *this;
}

int Chromosome::tovalue (int indGenInitial, int numGenes) const {
   int indGenFinal = indGenInitial + numGenes - 1;
   if (indGenFinal >= numTotalGenes) throw;

   int value = 0 ;
   int powerof2 = 1;
   for (int i=indGenInitial; i <= indGenFinal; i++){
      if (genes[i]) value += powerof2;
      powerof2 *= 2;
   }
   return value;
}

void Chromosome::mutate (){
   for (int i=0; i < numTotalGenes; i++){
      if ( flip (pMutate ) ) change (i);
   }
}
```

```
void Chromosome::change (int indGen){
   genes[indGen] = ! genes[indGen];
}


Parameter::Parameter ( int numG, double vMin, double vMax )
           :numGenes(numG), valueMin(vMin), valueMax(vMax) {
   int scale = 1;
   scale = scale << numGenes;  // 2 to the power of numGenes
   valueDecimalMax = scale - 1;
}

double Parameter::tovalue (int valueDecimal) const {
   // translates from the range [0, 2**L - 1] to the range [min, max]
   double range = valueMax - valueMin;
   return valueMin + ((double) valueDecimal) * range / valueDecimalMax;
}


// Parameters list
std::vector <Parameter> Parameters::Parameterslist;

void Parameters::addParameter ( int numGenes, double valueMin, double
                                                    valueMax ){
   Parameter param (numGenes, valueMin, valueMax);
   Parameterslist.push_back (param);
}

int Parameters::getNumTotalGenes (){
   int numGenes = 0;
   for (int i = 0; i < getNumParameters(); i++){
        numGenes += getParameter(i).getNumGenes();
   }
   return numGenes;
}
Phenotype::Phenotype ( const Chromosome & genotype ){
   int numParameters = Parameters::getNumParameters();
   values = new double [numParameters];
   int indGenInitial = 0;
   for (int i = 0; i < numParameters; i++){
      int numGenes = Parameters::getParameter(i).getNumGenes();
      int valueDecimal = genotype.tovalue (indGenInitial, numGenes);
      values[i] = Parameters::getParameter(i).tovalue(valueDecimal);
        indGenInitial += numGenes;
   }
}

Phenotype::Phenotype ( const Phenotype & f ){
   int numParameters = Parameters::getNumParameters();
   values = new double [numParameters];
   for (int i = 0; i < numParameters; i++) values[i] = f.values[i];
}
Phenotype & Phenotype::operator = ( const Phenotype & f ){
   if ( &f != this ){
      int numParameters = Parameters::getNumParameters();
      delete [] values;
      values = new double [numParameters];
      for (int i = 0; i < numParameters; i++) values[i] = f.values[i];
   }
   return *this;
```

```
}


const int Scale::K = 2;  // it must be >1
double Scale::a = 1;
double Scale::b = 0;

void Scale::recalculateScale ( double fitnessMin, double fitnessMax,
                                                  double fitnessAvg ){
   if (fitnessMin > ( (K*fitnessAvg-fitnessMax)/(K-1)) ){
      double delta = fitnessMax - fitnessAvg;
      a = (K-1) * fitnessAvg / delta;
      b = fitnessAvg * (fitnessMax - K * fitnessAvg) / delta;
   }
   else{
      double delta = fitnessAvg - fitnessMin;
      a = fitnessAvg / delta;
      b = - fitnessMin * fitnessAvg / delta;
   }
}

double Scale::getFitnessScaled ( double fitness ){
   return a * fitness + b;
}


Individual::Individual (){
   // create a random individual
   int numTotalGenes = Parameters::getNumTotalGenes();
   genotype = new Chromosome (numTotalGenes);
   phenotype = new Phenotype (*genotype);
   evaluate ();
}

Individual::Individual ( const Individual & father,
                                     const Individual & mother, int Crosspoint
){
   // Crosspoint = 0 --> first gen of father, the rest of mother
   // Crosspoint = numTotalGenes-1 --> all the gens of father
   genotype = new Chromosome (*father.genotype, *mother.genotype,
         Crosspoint);
   phenotype = new Phenotype (*genotype);
   evaluate ();
}

Individual::~Individual (){
   delete genotype;
   delete phenotype;
}

Individual::Individual ( const Individual & indiv ){
   genotype = new Chromosome (*indiv.genotype);
   phenotype = new Phenotype (*indiv.phenotype);
   fitnessReal = indiv.fitnessReal;
   fitnessofWork = indiv.fitnessofWork;
}

Individual & Individual::operator = ( const Individual & indiv ){
   if ( &indiv != this ){
      *genotype = *indiv.genotype;
```

```
      *phenotype = *indiv.phenotype;
      fitnessReal = indiv.fitnessReal;
      fitnessofWork = indiv.fitnessofWork;
   }
   return *this;
}

void Individual::evaluate (){
   // calculate fitness
   fitnessReal = externalEvaluation (*phenotype);
   fitnessofWork = 0.0;
}

void Individual::mutate (){
   genotype->mutate();
   delete phenotype;
   phenotype = new Phenotype (*genotype);
   evaluate ();
}

bool Individual::isValid () const {
   // it fulfills or not the constraints
   return isValidExternalSolution ( *phenotype );
}


std::ostream & operator<< (std::ostream & os, const Individual & indiv){
   int numParameters = Parameters::getNumParameters();
   os << "[";
   for (int i = 0; i < numParameters; i++){
      os << indiv.phenotype->getParameterValue (i) << "  ";
   }
   return os << "]" << std::endl;
}


SRemainder::SRemainder ( const std::vector <Individual> & Individualslist,
                                 double avgFitnessofWork ){
   // based on "stochastic remainder without replacement"

   numIndividuals = Individualslist.size();
   arrayIndexes = new int [numIndividuals];
   unSelected = numIndividuals;

   // fill the array
   // each element of the array is an index between 0 and numIndividuals-1

   double * fractions = new double [numIndividuals];
   int ind = 0;  // one index to the array of indexes
   // assign the integer number of copies
   for (int i = 0; i < numIndividuals; i++){
      double hopedOffsprings = Individualslist[i].getFitnessofWork() /
            avgFitnessofWork;
      int Fixedoffsprings = (int) floor (hopedOffsprings);
      fractions[i] = hopedOffsprings - (double) Fixedoffsprings;
      for (int j = 0; j < Fixedoffsprings; j++) arrayIndexes[ind++] = i;
   }
   // assign the fractional number of copies
   int i = 0;
   while (ind < numIndividuals){
```

```
      if ( fractions [i] > 0 ){
         bool win = flip ( fractions[i] );
         if (win){
            arrayIndexes[ind++] = i;
            // not to win again
            fractions[i] = 0;
         }
      }
      i++;
      if (i>=numIndividuals) i=0;
   }
   delete [] fractions;
}

SRemainder::~SRemainder (){
   delete [] arrayIndexes;
}

int SRemainder::getIndex (){
   // Give all the array indexes one by one. The already given
   // indexes are at the top and the rest at the bottom.

   // It gives a random index from the array of indexes between the
   // indexes not given yet.
   if ( unSelected == 0 ) throw;
   int sel = getRandomIndex (0, unSelected - 1);
   int result = arrayIndexes [sel];
   // move to the begining the last index.
   arrayIndexes[sel] = arrayIndexes[unSelected - 1];
   unSelected--;
   return result;
}
Population::Population (){
   sumFitnessReal = 0.0;
   maxFitnessReal = 0.0;
   minFitnessReal = DBL_MAX;
   sumFitnessofWork = 0.0;
   sremainder = 0;
}

Population::Population ( int numIndividuals ){
   sumFitnessReal = 0.0;
   maxFitnessReal = 0.0;
   minFitnessReal = DBL_MAX;
   for (int i = 0; i < numIndividuals; i++){
      Individual ind;   // random
      Individualslist.push_back (ind);
      double fit = ind.getFitnessReal();
      sumFitnessReal += fit;
      if (fit > maxFitnessReal) maxFitnessReal = fit;
      if (fit < minFitnessReal) minFitnessReal = fit;
   }
   std::sort ( Individualslist.begin(), Individualslist.end() );
   sremainder = 0;
   endofaGeneration ();  // calculate fitnessofWork
}

Population::~Population (){
   if (sremainder) delete sremainder;
}
```

```
void Population::addIndividual ( const Individual & ind ){
   Individualslist.push_back (ind);
   double fit = ind.getFitnessReal();
   sumFitnessReal += fit;
   if (fit > maxFitnessReal) maxFitnessReal = fit;
   if (fit < minFitnessReal) minFitnessReal = fit;
   std::sort ( Individualslist.begin(), Individualslist.end() );
}

void Population::endofaGeneration (){
   Scale::recalculateScale (minFitnessReal, maxFitnessReal,
         getFitnessRealAverage() );
   typedef std::vector<Individual>::iterator iter;
   sumFitnessofWork = 0.0;
   for (iter i = Individualslist.begin(); i !=
         Individualslist.end(); ++i ){
      i->updateFitnessofWork ();
      sumFitnessofWork += i->getFitnessofWork();
   }
}

const Individual & Population::select () const {
   // Stochastic Remainder method
   if (sremainder == 0 ) sremainder = new SRemainder ( Individualslist,
         getFitnessofWorkAverage() );
   return Individualslist[ sremainder->getIndex() ];
}


std::ostream & operator<< (std::ostream & os, const Population & pob){
   if ( ! pob.getBestIndividual ().isValid() ) os <<
         "this is not a feasible solution" << std::endl;
   os << "Best value = " << pob.getBestFitnessReal () << std::endl
      << "Mean value = " << pob.getFitnessRealAverage () << std::endl
      << "Parameters  = " << pob.getBestIndividual () << std::endl <<
std::endl;

   // To obtain the evolution graphics
   std::ofstream ofs ("priv1.txt", std::ios::app);
   ofs << pob.getBestFitnessReal () << "\t" << pob.getFitnessRealAverage ()
         << "\t" << pob.getBestIndividual () << std::endl;
   ofs.close();

   // To save a vector with 1 for feasible solution and 0 for the
   // non-feasible ones
   std::ofstream ovv ("validvector.txt", std::ios::app);
   if ( ! pob.getBestIndividual ().isValid() ) ovv << "0" << std::endl;
   if ( pob.getBestIndividual ().isValid() ) ovv << "1" << std::endl;
   ovv.close();
   return os;
}


GeneticAlgorithm::GeneticAlgorithm (){
   thePopulation = 0;
   theNewPopulation = 0;
   numGenerations = 0;
   showContinuous = false;
   numGenerationsMax = 0;
```

```
   minFitnessRealDesired = 0;
}

GeneticAlgorithm::~GeneticAlgorithm (){
   if (thePopulation) delete thePopulation;
   if (theNewPopulation) delete theNewPopulation;
}

void GeneticAlgorithm::initialize ( int numIndividuals, double pCr, double
pMt,
                                                    bool showCt, int ngMax,
double mf ){
   if ( Parameters::getNumParameters () == 0 ) throw;
   // even number of individuals
   if ( numIndividuals % 2 ) throw;
   // global variables
   numIndividualsInPopulation = numIndividuals;
   pMutate = pMt;
   pCross = pCr;
   // random numbers
   srand ( time (NULL) );
   // first generation
   thePopulation = new Population (numIndividuals);
   theNewPopulation = new Population ();
   numGenerations = 1;
   // member data
   showContinuous = showCt;
   numGenerationsMax = ngMax;
   minFitnessRealDesired = mf;
}

void GeneticAlgorithm::mainAlgorithm ( std::ostream & os ){
   while ( ! stop() ){

      // suppose a even number of individuals
      for ( int i = 0; i < numIndividualsInPopulation; i=i+2 ){
         const Individual & father = select();
         const Individual & mother = select();
         cross ( father, mother );  // mutation
      }
      if (showContinuous) show(os);
      passGeneration();
   }
   if (! showContinuous) show(os);
}

const Individual & GeneticAlgorithm::select (){
   return thePopulation->select ();
}

void GeneticAlgorithm::cross (const Individual & father,
                                              const Individual & mother){
   int numGenes = Parameters::getNumTotalGenes();
   int Crosspoint = numGenes - 1;
   // Crosspoint = 0 --> first gen of father, the rest of mother
   // Crosspoint = numTotalGenes-1 --> all the gens of father
   if ( flip (pCross) ) Crosspoint = getRandomIndex (0, numGenes - 2);
   // offsprings
   Individual offspring1 ( father, mother, Crosspoint );
   Individual offspring2 ( mother, father, Crosspoint );
```

```
   // mutation
   offspring1.mutate();
   offspring2.mutate();
   // save
   theNewPopulation->addIndividual ( offspring1 );
   theNewPopulation->addIndividual ( offspring2 );
}

void GeneticAlgorithm::mutate (Individual & indiv){
   indiv.mutate();
}

void GeneticAlgorithm::passGeneration (){
   delete thePopulation;
   thePopulation = theNewPopulation;
   thePopulation->endofaGeneration ();  // calculate fitnessofWork
   theNewPopulation = new Population ();
   numGenerations++;
}

bool GeneticAlgorithm::stop (){
   return (numGenerations >= numGenerationsMax) ||
          (thePopulation->getBestFitnessReal () >= minFitnessRealDesired);
}

void GeneticAlgorithm::show ( std::ostream & os ){
   os << "Generation " << numGenerations << std::endl << *thePopulation;
}


// BOUNDS.h       Classes definition to represent the QFT bounds and to
//                calculate the cost of a controller. The fitness function
//                is based on Thompson [39].
//
// Authors: Mario Garcia-Sanz, Carlos Molins
// 1.September.2009
//-------------------------------------------------------------------------

#if !defined __BOUNDS_H__
#define __BOUNDS_H__

#include <complex>
#include <vector>
#include <algorithm>       // for sort()
#include <iostream>
#include "aagg.h"


class qcomplex {

   private:

      std::complex<double> c;

   public:

      qcomplex (double re = 0, double im = 0) : c(re,im){}
      qcomplex (std::complex<double> cc) : c(cc){}
      ~qcomplex (){}
```

```
      // qcomplex ( const qcomplex & );
      // qcomplex & operator = ( const qcomplex & );

      double real() const { return c.real(); }
      double imag() const { return c.imag(); }
      double abs() const { return std::abs(c); }
      double arg() const { return std::arg(c); }
      qcomplex conj() const { return std::conj(c); }

      operator std::complex<double> () { return c; }
      operator std::complex<double> () const { return c; }

      // to keep arranged the zeros of a polynomial
      // from lower to higher, with the most significative at the
        // beginning.
      bool operator< ( const qcomplex & c2 ) const {
              return real() < c2.real(); }
      bool operator== ( const qcomplex & c2 ) const {
              return real() == c2.real() && imag() == c2.imag(); }
      friend std::istream & operator>> (std::istream & is,
              qcomplex & qc);
};


std::istream & operator>> (std::istream & is, qcomplex & qc){
   return is >> qc.c;
}

// Represents a numerator or a denominator of a rational transfer
// function in the form (s+z1)(s+z2)... where z1, z2, ... may be
// complex.
//
// The list is kept arranged from the most to the less sinificative
// in order to select the first n zeros.

class Polynom {

      std::vector <qcomplex> Zeroslist;
      int numOriginZeros;

   public:

         Polynom (){ numOriginZeros = 0; }

      // By default:
      // Polynom ( const Polynom & );
      // Polynom & operator = ( const Polynom & );

      void addZero ( qcomplex c );
      int getNumZeros () const { return Zeroslist.size(); }
      int getNumZerosNotZero () const {
              return Zeroslist.size() - numOriginZeros; }
      qcomplex getZero (int i) const { return Zeroslist[i]; }
      qcomplex getZeroNotZero (int i) const {
              return Zeroslist[i+numOriginZeros]; }
      qcomplex evaluateIn ( double frequency ) const;

      friend Polynom operator * ( const Polynom & p1,
              const Polynom & p2 );
};
```

```
Polynom operator * ( const Polynom & p1,
                           const Polynom & p2 );


// Represents a rational transfer function in the form
//        Kg(s+z1)(s+z2).../(s+p1)(s+p2)...

class FT {

      double Kg;       // high frequency gain.
      Polynom numerator;
      Polynom denominator;

   public:

      FT (){}

      // By default:
      // FT ( const FT & );
      // FT & operator = ( const FT & );

      int getNumZeros () const { return numerator.getNumZeros(); }
      int getNumZerosNotZero () const {
              return numerator.getNumZerosNotZero(); }
      int getNumPolos () const { return denominator.getNumZeros(); }
      int getNumPolesNotZero () const {
              return denominator.getNumZerosNotZero(); }
      qcomplex getZero (int i) const {
              return numerator.getZero(i); }
      qcomplex getZeroNotZero (int i) const {
              return numerator.getZeroNotZero(i); }
      qcomplex getPole (int i) const {
              return denominator.getZero(i); }
      qcomplex getPoleNotZero (int i) const {
              return denominator.getZeroNotZero(i); }
      void addZero ( qcomplex z ){ numerator.addZero(z); }
      void addPole ( qcomplex p ){ denominator.addZero(p); }
      void setKg ( double k ){ Kg = k; }
      double getKg () const { return Kg; }
      qcomplex evaluateIn ( double frec ) const {
              return (std::complex<double>) numerator.evaluateIn(frec) * Kg /
                     (std::complex<double>) denominator.evaluateIn(frec); }
      double getArea (double wIni, double wEnd) const;
      friend FT operator * ( const FT & ft1, const FT & ft2 );
};


FT operator * ( const FT & ft1, const FT & ft2 );


// Represents a bound at a certain frequency:
//
// It is a double set of values, with the maximum and the minimum
// permitted values for each phase value.

class Bound {
```

```
   private:

      double frequency;
      double numValues;   // phase points in which the bound has been
                                    // evaluated
      double * minValues; // bounds in solid line in QFT of MATLAB
      double * maxValues; // bounds in dashed line in QFT of MATLAB

      // NOTE: there is no relation between minValues and maxValues; i.e.,
        //       it may happen minValues[i]>maxValues[i]

      double getDeltaPhase() const { return 360/(numValues-1); }
      // gives the bound at a certain phase (extrapolation if necessary)
      void getBoundsForPhase( double phase, double & min, double & max )
             const;

   public:

      Bound (){ minValues = maxValues = 0; }  // necessary to use later
                                                                    //
std::vector
      Bound ( std::istream & is );
      ~Bound ();

      Bound ( const Bound & );
      Bound & operator = ( const Bound & );

      double getFrequency() const { return frequency; }
      double getCost ( const FT & ftla ) const;

      // necessary to use later std::vector
      bool operator< ( const Bound & b ) const { return true; }
      bool operator== ( const Bound & b ) const { return false; }
};



// Set of bounds

class Bounds {

      std::vector <Bound> Boundslist;

      Bounds ( const Bounds & );
      Bounds & operator = ( const Bounds & );
   public:

      Bounds ( std::istream & is );

      int getNumBounds () const { return Boundslist.size(); }
      double getCostDueToBound ( int i, const FT & ftla ) const;

};



// Class to calculate the cost of a controller. This cost has two
// components:
//   - basic cost (function to minimize)
```

```
//   - penalty (contraints that have to be fulfilled)

class FCost {

      double a0, a1, a2;   // basic cost function parameters
      double w1, w2;       // interval of frequencies to measure the
                                    // controller area
      double b;            // penalty parameter
      Bounds * theBounds;  // QFT bounds
      FT thePlant;         // QFT nominal plant

      FCost ( const FCost & );
      FCost & operator = ( const FCost & );

      double getBasicCost ( const FT & controller ) const;
      double getPenalty ( const FT & controller ) const;

   public:

      FCost ();
      ~FCost ();

      bool fulfilBounds ( const FT & contr ) const {
              return getPenalty(contr) == 0; }
      double getCost ( const FT & contr ) const {
              return getBasicCost(contr) + getPenalty(contr); }
};



// Utility class

class XContr {

      // number of controller fixed poles, i.e., the algorithm
        // does not move them
      int numFixedPoles;
      double * theFixedPoles;
      // number of parameters that corresponds to the controller
        // zeros
      int numZeros;

      // the parameters must be:
      //   - the first is the controller Kg
      //   - the following "numZeros" are the controller zeros
      //   - the rest are the controller poles
      XContr ( const XContr & );
      XContr & operator = ( const XContr & );

   public:

      XContr ();
      ~XContr ();

      FT getController ( const Phenotype & phenotype );

};

#endif
```

```
// BOUNDS.cpp     QFT bounds representation and controller cost calculation.
//
// Authors: Mario Garcia-Sanz, Carlos Molins
// 1.September.2009
//-------------------------------------------------------------------------


#include <math.h>        // for floor()
#include <strstream>
#include <fstream>
#include "bounds.h"


#define M_PI        3.14159265358979323846


// Function to obtain dB and phase
void dbphase ( qcomplex c, double & db, double & phase ){
   db=20*log10( c.abs() );
   phase=180*c.arg()/M_PI;
   // to transform from (-180,180] to (-360,0]
   if (phase > 0) phase -= 360.0;
}


void Polynom::addZero ( qcomplex c ){
   Zeroslist.push_back (c);
   // add the conjugate
   if ( c.imag() ) Zeroslist.push_back (c.conj());
   std::sort ( Zeroslist.begin(), Zeroslist.end() );
   if ( (c.real() == 0) && (c.imag() == 0) ) numOriginZeros++;
}

qcomplex Polynom::evaluateIn ( double frequency ) const {
   std::complex<double> jw (0, frequency);
   std::complex<double> res (1);  // initialize to the unity
   for (int i=0; i<Zeroslist.size(); i++){
      res *= (jw+(std::complex<double>)Zeroslist[i]);
   }
   return res;
}


Polynom operator * ( const Polynom & p1, const Polynom & p2 ){
   Polynom res;
   for (int i=0; i<p1.Zeroslist.size(); i++) res.Zeroslist.push_back
         (p1.Zeroslist[i]);
   for (int i=0; i<p2.Zeroslist.size(); i++) res.Zeroslist.push_back
         (p2.Zeroslist[i]);
   return res;
}


double FT::getArea (double wIni, double wEnd) const {
   // area of ln|G(jw)|
   double area = 0;
   const int NUMW = 100;
   double deltaw = (wEnd - wIni) / NUMW;
```

```
   double w = wIni;
   for (int i=0; i<NUMW; i++){
      qcomplex g = evaluateIn (w);
      area += deltaw * log (g.abs());
      w += deltaw;
   }
   return area > 0 ? area : 0.0;
}


FT operator * ( const FT & ft1, const FT & ft2 ){
   FT res;
   res.setKg ( ft1.Kg * ft2.Kg );
   res.numerator = ft1.numerator * ft2.numerator;
   res.denominator = ft1.denominator * ft2.denominator;
   return res;
}


Bound::Bound ( std::istream & is ){
   is >> frequency >> numValues;
   minValues = new double [numValues];
   maxValues = new double [numValues];
   for (int i=0; i<numValues; i++) is >> minValues[i];
   for (int i=0; i<numValues; i++) is >> maxValues[i];
}

Bound::~Bound (){
   if (minValues) delete [] minValues;
   if (maxValues) delete [] maxValues;
}

Bound::Bound ( const Bound & bnd ){
   frequency = bnd.frequency;
   numValues = bnd.numValues;
   minValues = new double [numValues];
   maxValues = new double [numValues];
   for (int i=0; i<numValues; i++){
      minValues[i] = bnd.minValues[i];
      maxValues[i] = bnd.maxValues[i];
   }
}

Bound & Bound::operator = ( const Bound & bnd ){
   if ( &bnd != this ){
      if (minValues) delete [] minValues;
      if (maxValues) delete [] maxValues;
      frequency = bnd.frequency;
      numValues = bnd.numValues;
      minValues = new double [numValues];
      maxValues = new double [numValues];
      for (int i=0; i<numValues; i++){
         minValues[i] = bnd.minValues[i];
         maxValues[i] = bnd.maxValues[i];
      }
   }
   return *this;
}

double Bound::getCost ( const FT & ftla ) const {
```

```
   qcomplex c = ftla.evaluateIn ( getFrequency() );
   double db, phase;
   dbphase ( c, db, phase );
   double min, max;
   getBoundsForPhase( phase, min, max );
   if ( max > min ){
      if ( db < min ) return min - db;
      if ( db > max ) return db - max;
   }
   else{
         if ( (db < min) && (db > max) )
               return (min-db)>(db-max) ? db-max : min-db;
   }
   return 0.0;
}

void Bound::getBoundsForPhase( double phase, double & min,
                                              double & max ) const {
   // linear interpolation
   if ( (phase>0) || (phase<-360) ) throw;
   double delta = - getDeltaPhase();
   // the bounds are for 0:-delta:-360 (MATLAB notation)
   double mudo = phase / delta;
   int ind = (int) floor (mudo);
   if (ind == numValues-1) ind--;
   //   x-x1/x2-x1 = y-y1/y2-y1
   //   point1 corresponds to ind, point2 to ind+1
   min = ( (minValues[ind+1] - minValues[ind])*(phase-ind*delta)/delta )
         + minValues[ind];
   max = ( (maxValues[ind+1] - maxValues[ind])*(phase-ind*delta)/delta )
         + maxValues[ind];
}


Bounds::Bounds ( std::istream & is ){
   char cadena [30000];

   while (!is.eof()){
      is.getline(cadena, sizeof(cadena));
      std::istrstream iss (cadena, sizeof (cadena) );
      Bound bnd (iss);
      if ( iss.good() ){
         Boundslist.push_back (bnd);
      }
   }
}
double Bounds::getCostDueToBound ( int i, const FT & ftla ) const {
   return Boundslist[i].getCost (ftla);
}


FCost::FCost (){
   std::ifstream ifsConf ("cfgcost.txt");
   if ( ! ifsConf.good() ){
      std::cerr << "\tCan not open the file to configure the fitness function"
                << std::endl;
      throw;
   }

   char cad [250];
```

```
   char ficBounds [250];
   ifsConf >> ficBounds;
   ifsConf.getline (cad, sizeof(cad));

   ifsConf >> a0;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> a1;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> a2;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> w1;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> w2;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> b;
   ifsConf.getline (cad, sizeof(cad));

   double k;
   ifsConf >> k;
   ifsConf.getline (cad, sizeof(cad));
   thePlant.setKg ( k );

   int numZeros;
   ifsConf >> numZeros;
   ifsConf.getline (cad, sizeof(cad));
   for (int i = 0; i < numZeros; i++){
      qcomplex zero;
      ifsConf >> zero;
      ifsConf.getline (cad, sizeof(cad));
      thePlant.addZero ( zero );
   }

   int numPoles;
   ifsConf >> numPoles;
   ifsConf.getline (cad, sizeof(cad));
   for (int i = 0; i < numPoles; i++){
      qcomplex pole;
      ifsConf >> pole;
      ifsConf.getline (cad, sizeof(cad));
      thePlant.addPole ( pole );
   }

   std::ifstream ifs (ficBounds);
   if ( ! ifs.good() ){
      std::cerr << "\tCan not open the bounds file" << std::endl;
      throw;
   }

   theBounds = new Bounds (ifs);
}

FCost::~FCost (){
   delete theBounds;
}

double FCost::getBasicCost ( const FT & controller ) const {
   // Fitness function based on Thompson and Nwokah[38] and Thompson [39]
   double cost0 = a0 * controller.getKg() * controller.getKg();
   // for the cost1 remove the less significative poles
```

```
   // until np=nz
   double c = 0;
   // cost1 takes into account poles and zeros different from 0

   int numZeros = controller.getNumZerosNotZero();
   if (controller.getNumPolesNotZero() < numZeros) numZeros =
         controller.getNumPolesNotZero();
   for (int i = 0; i < numZeros; i++){
      // It is supossed that the controller has no complex zeros and no
        // complex poles.
      qcomplex pi = controller.getPoleNotZero(i);
      qcomplex zi = controller.getZeroNotZero(i);
      double dif = pi.real()-zi.real();
      c += ( (dif * dif * dif * dif) + 1 ) / (pi.real() * zi.real());
   }
   double cost1 = a1 * c;
   double cost2 = a2 * controller.getArea(w1, w2);

   return cost0 + cost1 + cost2;
}

double FCost::getPenalty ( const FT & controller ) const {
   double cost = 0;
   FT ftla = controller * thePlant;
   int numBounds = theBounds->getNumBounds ();
   for (int i=0; i<numBounds; i++){
      double c = theBounds->getCostDueToBound ( i, ftla );
        cost += b * c * c;
   }

   return cost;
}


XContr::XContr (){
   std::ifstream ifsConf ("cfgcontr.txt");
   if ( ! ifsConf.good() ){
      std::cerr << "\tCan not open the controller configuration file"
              << std::endl;
      throw;
   }

   char cad [250];

   ifsConf >> numFixedPoles;
   ifsConf.getline (cad, sizeof(cad));
   theFixedPoles = new double [numFixedPoles];
   for (int i = 0; i < numFixedPoles; i++){
      ifsConf >> theFixedPoles[i];
      ifsConf.getline (cad, sizeof(cad));
   }
   ifsConf >> numZeros;
   ifsConf.getline (cad, sizeof(cad));
}

XContr::~XContr (){
   delete [] theFixedPoles;
}

FT XContr::getController ( const Phenotype & phenotype ){
```

```
   // NOTE: the number of poles different from 0 to optimize hve to be
   // the same as the number of zeros different from 0 to optimize.The
   // remainder poles or zeros have to be fixed (Thompson)

   FT controller;
   controller.setKg ( phenotype.getParameterValue (0) );
   for (int i = 1; i <= numZeros; i++){
      controller.addZero ( (qcomplex) phenotype.getParameterValue (i) );
   }
   int numParameters = Parameters::getNumParameters();
   for (int i = numZeros+1; i < numParameters; i++){
      controller.addPole ( (qcomplex) phenotype.getParameterValue (i) );
   }
   for (int i = 0; i < numFixedPoles; i++){
      controller.addPole ( (qcomplex) theFixedPoles[i] );
   }
   return controller;
}


static FCost fcost;
static XContr xcontr;


// Fitness function
double externalEvaluation ( const Phenotype & phenotype ){
   FT controller = xcontr.getController (phenotype);
   // the objective of the Genetic Algorithms is to find the maximum
   return 1/fcost.getCost( controller );
}

// Function to check the feasibility
// (no constraint violation)
bool isValidExternalSolution ( const Phenotype & phenotype ){
   // if no constraint:
   // return true;

   // If there are constraints:
   FT controller = xcontr.getController (phenotype);
   return fcost.fulfilBounds( controller );
}




// OPTQFT.cpp     This is the main program that tries to obtain an optimal
//                      controller applying the Genetic Algorithms.
//
// Authors: Mario Garcia-Sanz, Carlos Molins
// 1.September.2009
//-------------------------------------------------------------------------

#include <math.h>
#include <iostream>
#include <fstream>
#include "aagg.h"


using namespace std;
```

```
void main( int argc, char * argv [] ){
   // Read configuration
   ifstream ifsConf ("cfg.txt");
   if ( ! ifsConf.good() ){
      cerr << "\tCan not open the configuration file" << endl;
      throw;
   }

   char cad [250];

   int numParameters;
   ifsConf >> numParameters;
   ifsConf.getline (cad, sizeof(cad));

   double * minParam = new double [numParameters];
   for (int i = 0; i < numParameters; i++){
      ifsConf >> minParam[i];
      ifsConf.getline (cad, sizeof(cad));
   }
   double * maxParam = new double [numParameters];
   for (int i = 0; i < numParameters; i++){
      ifsConf >> maxParam[i];
      ifsConf.getline (cad, sizeof(cad));
   }

   double precision;
   ifsConf >> precision;
   ifsConf.getline (cad, sizeof(cad));

   int numIndividuals;
   double pCross, pMutate;
   int showContinuous;
   int maxGenerations;
   double minFitness;

   ifsConf >> numIndividuals;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> pCross;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> pMutate;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> showContinuous;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> maxGenerations;
   ifsConf.getline (cad, sizeof(cad));
   ifsConf >> minFitness;
   ifsConf.getline (cad, sizeof(cad));

   ofstream ofs ("aagg.log");
   if ( ! ofs.good() ){
      cerr << "\tCan not open the output file" << endl;
      return;
   }

   // Initialize the parameters: numGenes, valueMin, valueMax

   // precision:
   //
```

```
   // The precision is calculated by:
   //   r=2**numGenes -1
   //   prec=(max-min)/r

   for (int i = 0; i < numParameters; i++){
      double y = ((maxParam[i]-minParam[i])/precision)+1;
      int numGenes = floor ( log((double)y)/log((double)2) ) + 1;
      Parameters::addParameter ( numGenes, minParam[i], maxParam[i] );
   }

   // Remove the file to obtain evolution graphics
   std::ofstream ofsPriv ("priv1.txt");
   ofsPriv.close();

   // Initialize the algorithm
   GeneticAlgorithm ag;
   ag.initialize ( numIndividuals, pCross, pMutate, (bool) showContinuous,
         maxGenerations, minFitness );

   // execute
   ag.mainAlgorithm ( ofs );

   delete [] minParam;
   delete [] maxParam;
}
```

```
// RANDOM.h       Functions with random numbers.C++ adaptation of the
//                      Goldberg basic code [23].
//
//
// Authors: Mario Garcia-Sanz, Carlos Molins
// 1.September.2009
//-------------------------------------------------------------------------


#if !defined __RANDOM_H__
#define __RANDOM_H__

// random number between 1 and 0
double rand01 ();

// gives TRUE with probability p
bool flip ( double p);

// gives a random index (integer) between min and max
int getRandomIndex (int min, int max);

#endif
```

```
// RANDOM.cpp     Functions with random numbers.C++ adaptation of the
//                Goldberg basic code [23].
//
//
// Authors: Mario Garcia-Sanz, Carlos Molins
// 1.September.2009
//-------------------------------------------------------------------------
```

```
#include <stdlib.h>
#include "random.h"


double rand01 (){
   // random number between 0 and 1
   int r = rand();
   return (double) r / (double) RAND_MAX;
}

bool flip (double p){
   // gives TRUE with probability p
   return rand01() <= p;
}

int getRandomIndex (int min, int max){
   // gives a random index (integer) between min and max

   int range = max - min + 1;
   int newMax = (long) RAND_MAX + 1 - ( (long) RAND_MAX + 1) % range;
   int temp = rand();
   while (temp > newMax ) temp = rand();
   return temp%range + min;
}
```

8.3. TEXT FILES

CFG.txt This file is to define range of variation of controller parameters, as well as the crossover and mutation probabilities, maximum number of generations, the number of individuals in each generation,…

```
11       number of parameters
0.01     minimum value for the parameter 1
0.01     minimum value for the parameter 2
0.01     minimum value for the parameter 3
0.01     minimum value for the parameter 4
0.01     minimum value for the parameter 5
0.01     minimum value for the parameter 6
0.01    minimum value for the parameter 7
0.01    minimum value for the parameter 8
0.01    minimum value for the parameter 9
0.01    minimum value for the parameter 10
0.01    minimum value for the parameter 11
1000    maximum value for the parameter 1
5       maximum value for the parameter 2
5       maximum value for the parameter 3
5       maximum value for the parameter 4
10      maximum value for the parameter 5
10      maximum value for the parameter 6
1000    maximum value for the parameter 7
1000    maximum value for the parameter 8
1000    maximum value for the parameter 9
1000    maximum value for the parameter 10
1000    maximum value for the parameter 11
.01     parameters precision
100     number of individuals
0.1     crossover probability
0.01    mutation probability
1       show continuously (1) or only (0)
500     maximum number of generations
100000  minimum desired fitness
```

CFGCONTR.txt This file is to define some controller characteristics.

```
0       number of controller fixed poles
5       number of parameters that corresponds to controller zeros
```

CFGCOST.txt This file is to define the fitness function parameters values, as well as the nominal plant.

```
bnd_n      bounds file
0.001      a0 parameter
0.00001    a1 parameter
1          a2 parameter
0.01       wIni to measure the "excess controller bandwidth"
1          wEnd to measure the "excess controller bandwidth"
1          a3 parameter
1440       gain of the QFT nominal plant
```

```
1          number of zeros of the QFT nominal plant
0.5        nominal plant zero; for complex: (2,3) only one of the conjugates
3          number of poles of the QFT nominal plant (without conjugates)
0          nominal plant pole; for complex: (2,3) only one of the conjugates
10         nominal plant pole; for complex: (2,3) only one of the conjugates
(4.8,3.6)  nominal plant pole; for complex: (2,3) only one of the conjugates
```

8.4. HOW TO RUN

First of all, move all the .cpp and .h files to the folder:

C:\Program Files\Microsoft Visual Studio 9.0\VC

Once this is done, write at the Visual Studio 2008 Command Prompt:

cl –EHsc OPTQFT.cpp aagg.cpp random.cpp bounds.cpp

With this you get: optqft.exe

Then, move this optqft.exe to the folder where the Matlab and text files are.

Finally, read the files just in case any other configuration is desired, and run at the Matlab Command Window: Autoloopshaping_AG.m